LECTURE 04

DIVIDE AND CONQUER

C++ python™ Java

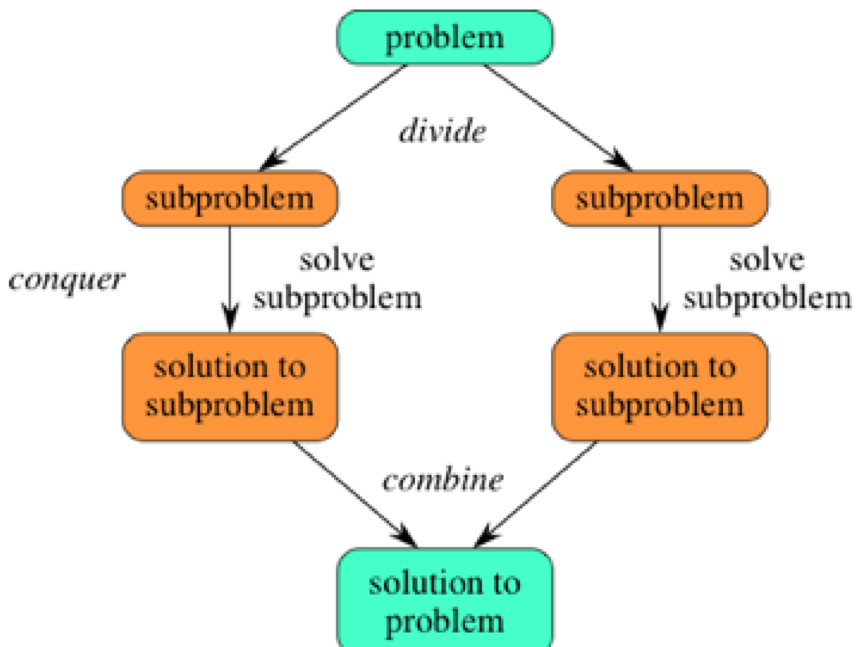
Big-O Coding

Website: www.bigocoding.com

Giới thiệu tổng quan

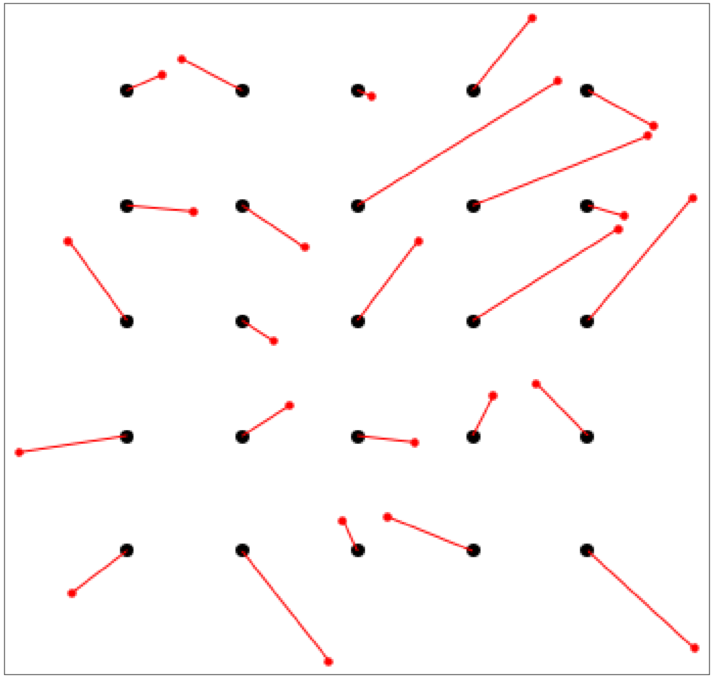
Divide and Conquer (Chia để trị):

- **Divide:** **Chia** bài toán lớn thành các bài toán con nhỏ hơn.
- **Conquer:** Sử dụng tính chất đệ quy để **trị** (giải) các bài toán con.
- **Combine:** **Ghép** lời giải của các bài toán con, để giải bài toán lớn ban đầu.



Bài toán minh họa 1

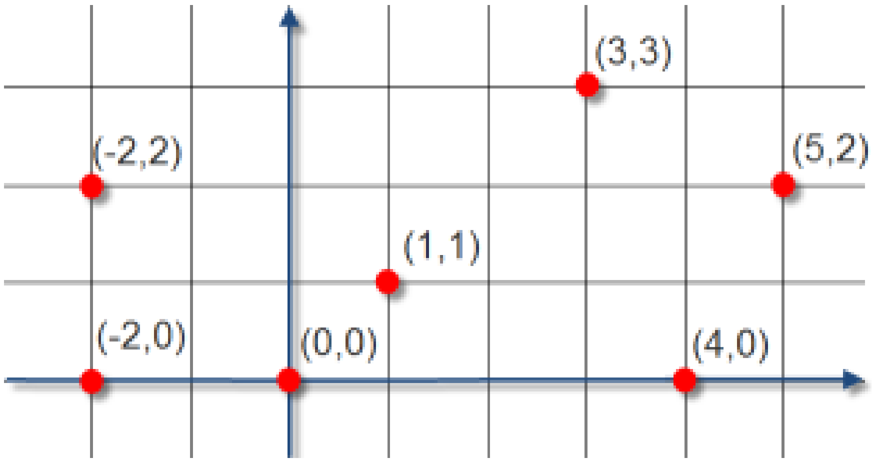
Closest Pair of Points: Cho N điểm trong mặt phẳng (các điểm không trùng nhau). Hãy tìm cặp điểm có khoảng cách gần nhau nhất.



Phương pháp giải quyết thông thường

Dùng phương pháp **Brute force**, duyệt qua từng cặp điểm và tính khoảng cách của mỗi cặp điểm. Cặp điểm nào có độ dài ngắn nhất là cặp điểm cần tìm.

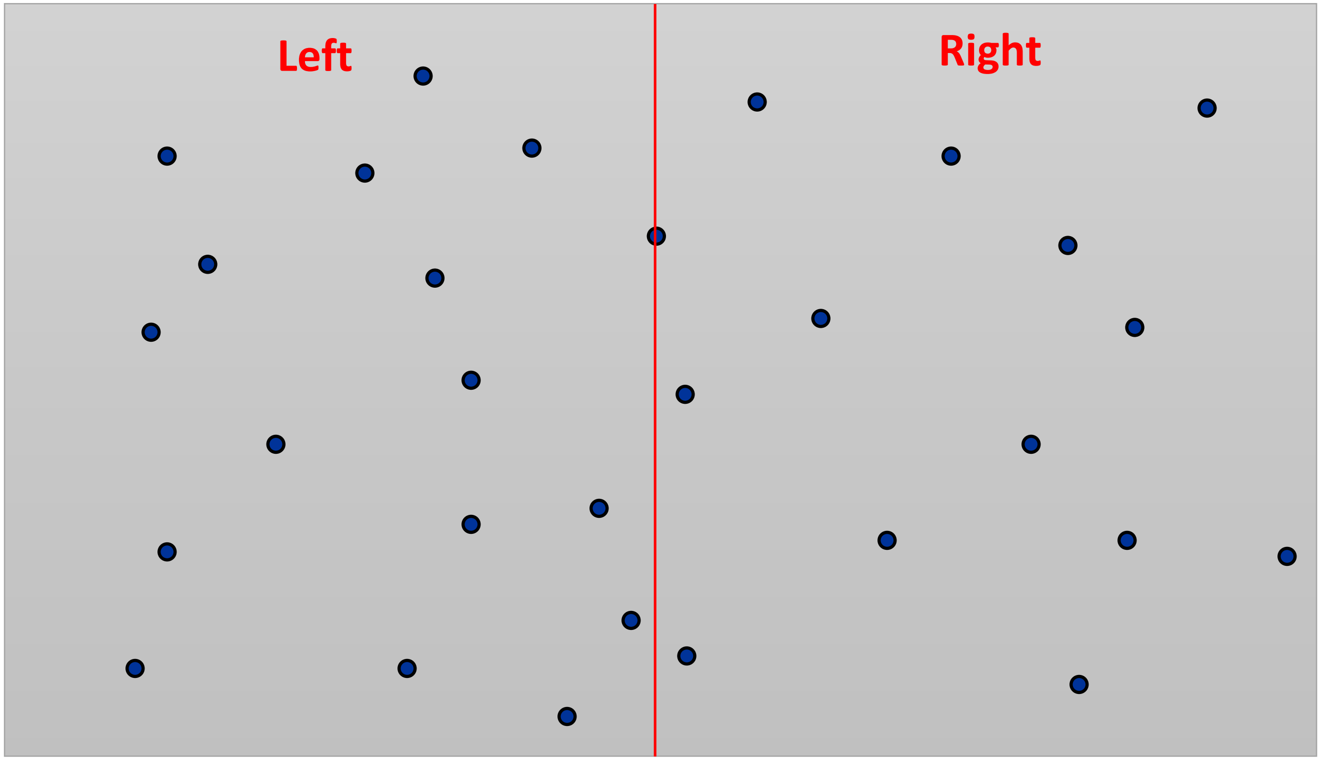
Công thức tính khoảng cách d = $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.



Độ phức tạp: **$O(N^2)$**

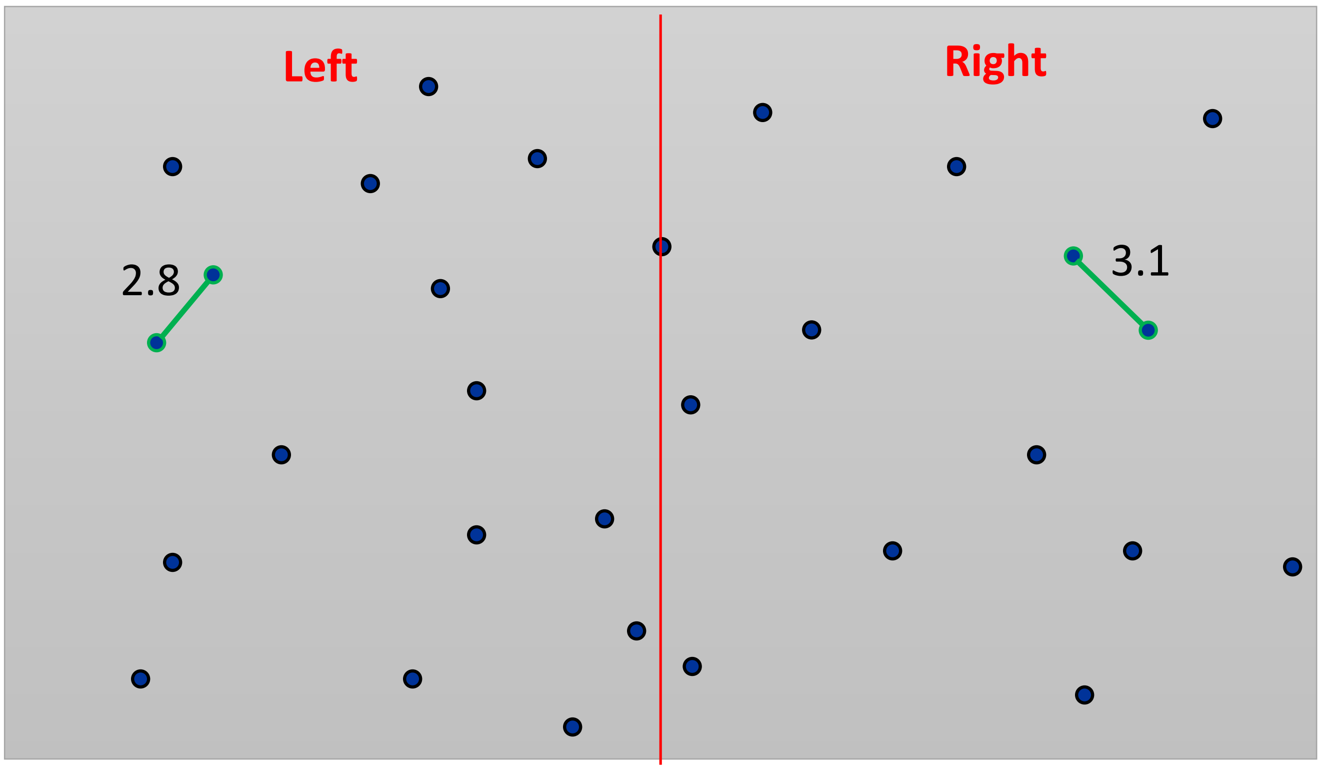
Phương pháp chia để trị

- **Divide:** **Chia** bài toán lớn thành các bài toán con nhỏ hơn.



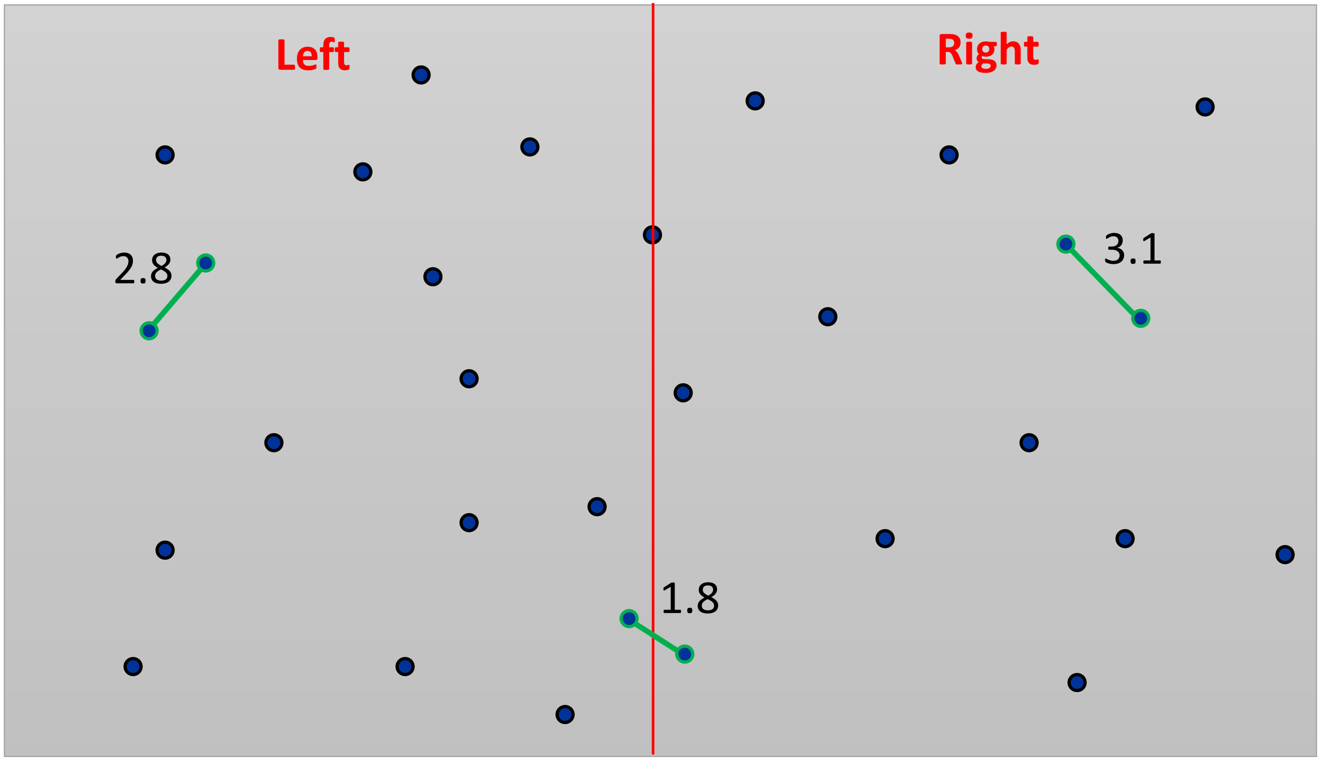
Phương pháp chia để trị

- **Divide:** **Chia** bài toán lớn thành các bài toán con nhỏ hơn.
- **Conquer:** Sử dụng tính chất đệ quy để **trị** (giải) các bài toán con.



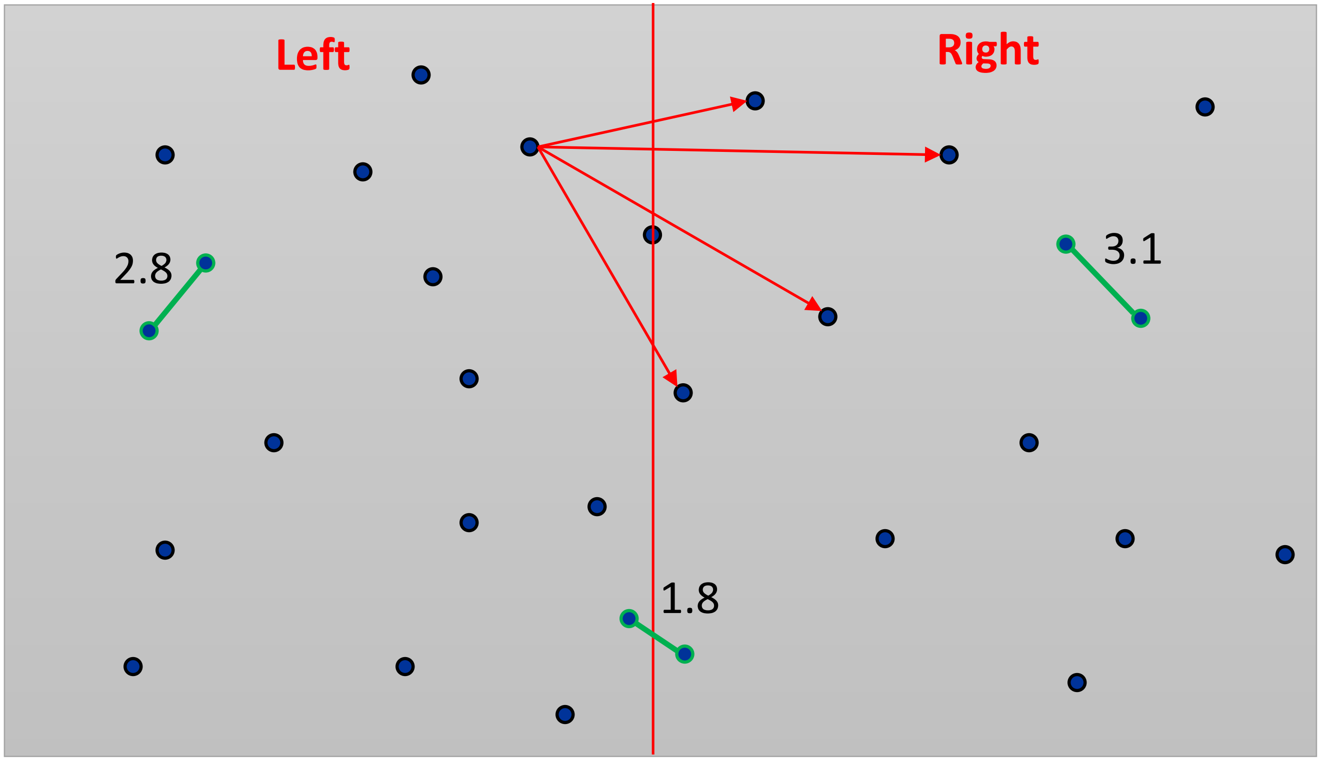
Phương pháp chia để trị

- **Divide:** **Chia** bài toán lớn thành các bài toán con nhỏ hơn.
- **Conquer:** Sử dụng tính chất đệ quy để **trị** (giải) các bài toán con.
- **Combine:** **Ghép** lời giải của các bài toán con, để giải bài toán lớn ban đầu.



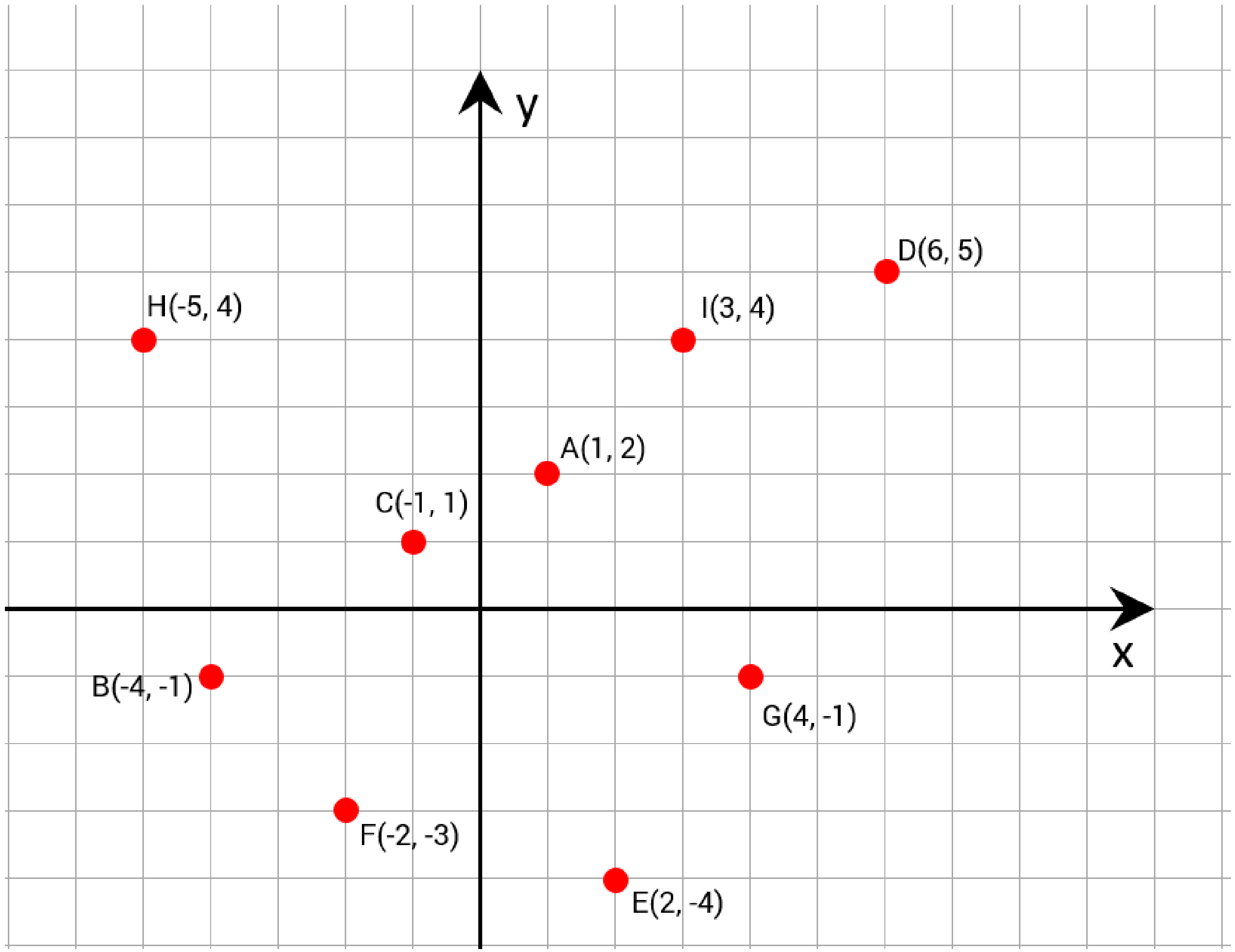
Phương pháp chia để trị

- Trường hợp khó khăn khi **Combine** là tính khoảng cách giữa các điểm thuộc 2 phía **Left, Right** có thể làm độ phức tạp của bài toán tăng lên rất lớn ➔ **Tìm giải pháp phù hợp.**



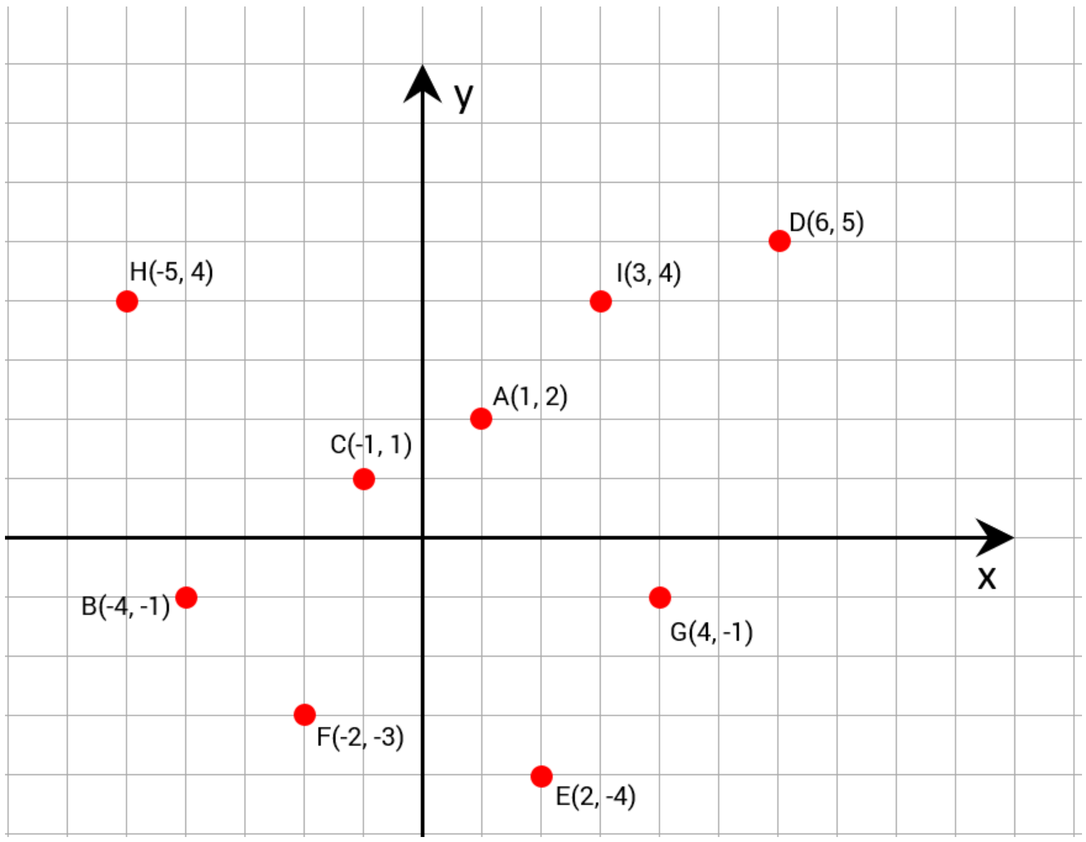
Bài toán minh họa 1

Ví dụ: Cho 9 điểm trên mặt phẳng tọa độ Oxy (hình vẽ), tìm cặp điểm có khoảng cách gần nhất:



Bước 0: Chuẩn bị dữ liệu

Từ dữ liệu đầu vào là danh sách tọa độ các điểm. Đọc dữ liệu vào mảng một chiều chứa danh sách các điểm.



Points List

```
9
1 2
-4 -1
-1 1
6 5
2 -4
-2 -3
4 -1
-5 4
3 4
```

Bước 1: Sắp xếp tọa độ điểm tăng dần theo x

Danh sách các tọa độ ban đầu trong mảng.

Tên	A	B	C	D	E	F	G	H	I
Tọa độ	(1, 2)	(-4, -1)	(-1, 1)	(6, 5)	(2, -4)	(-2, -3)	(4, -1)	(-5, 4)	(3, 4)

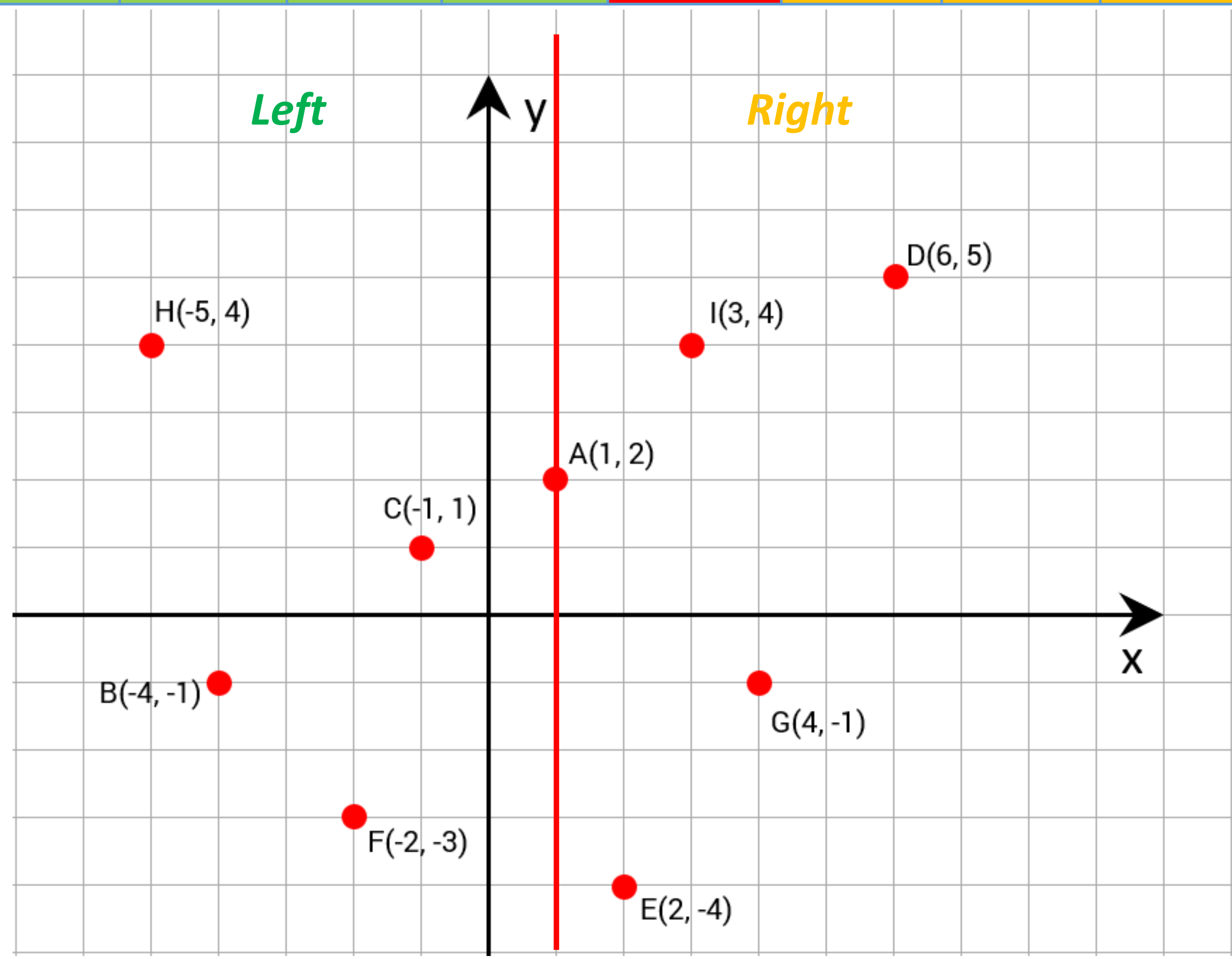
Sắp xếp tọa độ điểm tăng dần theo **hoành độ x**.

Tên	H	B	F	C	A	E	I	G	D
Tọa độ	(-5, 4)	(-4, -1)	(-2, -3)	(-1, 1)	(1, 2)	(2, -4)	(3, 4)	(4, -1)	(6, 5)

Bước 2: Chia mảng thành 2 phần

Chia các tọa độ điểm làm 2 phần left – right.

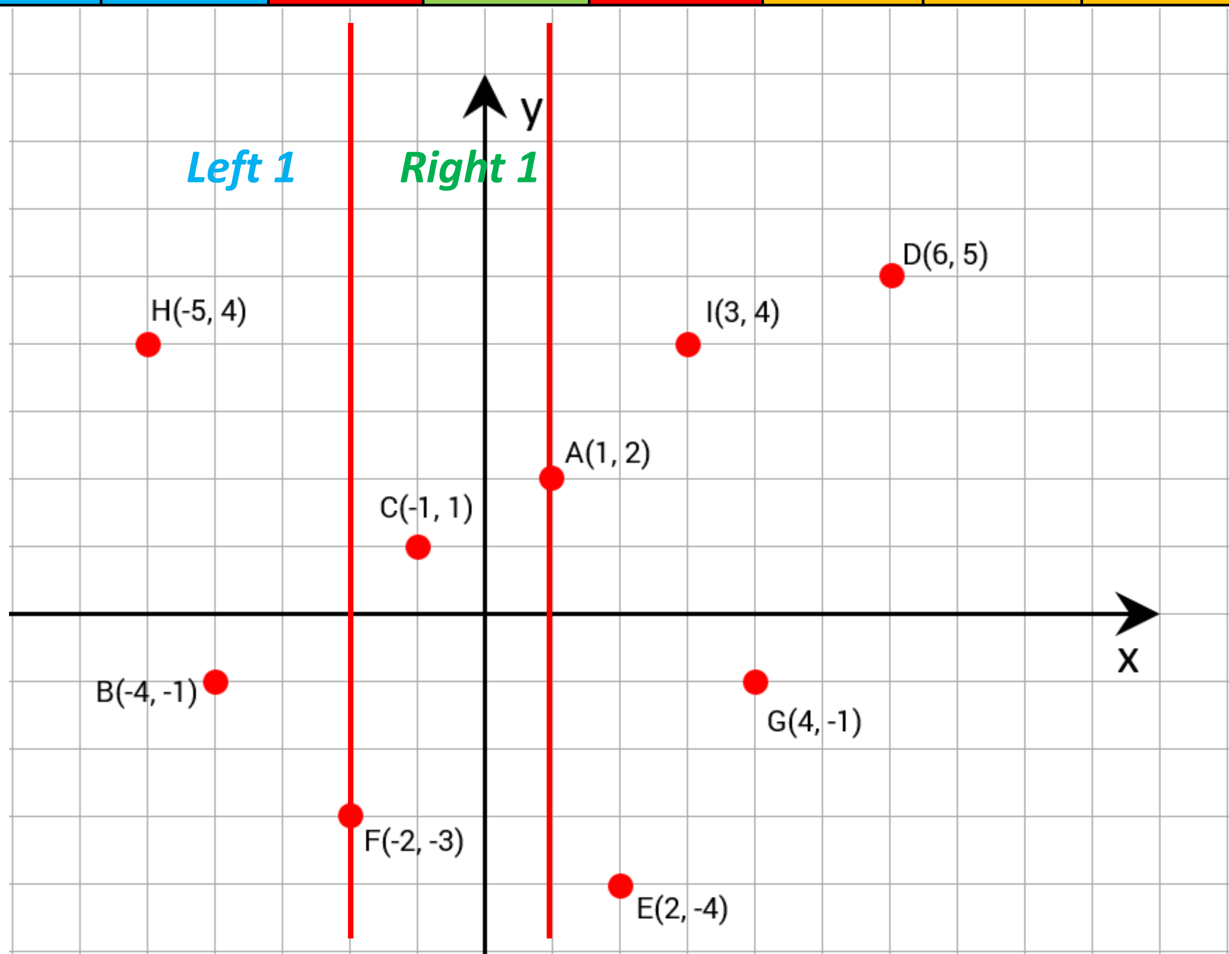
Tên	H	B	F	C	A	E	I	G	D
Tọa độ	(-5, 4)	(-4, -1)	(-2, -3)	(-1, 1)	(1, 2)	(2, -4)	(3, 4)	(4, -1)	(6, 5)



Bước 2.1: Chia mảng Left thành 2 phần

Do 2 phần ở bước trên chưa đủ nhỏ nên tiếp tục chia đoạn bên left ra 2 phần nhỏ hơn.

Tên	H	B	F	C	A	E	I	G	D
Tọa độ	(-5, 4)	(-4, -1)	(-2, -3)	(-1, 1)	(1, 2)	(2, -4)	(3, 4)	(4, -1)	(6, 5)



Big-O Orange

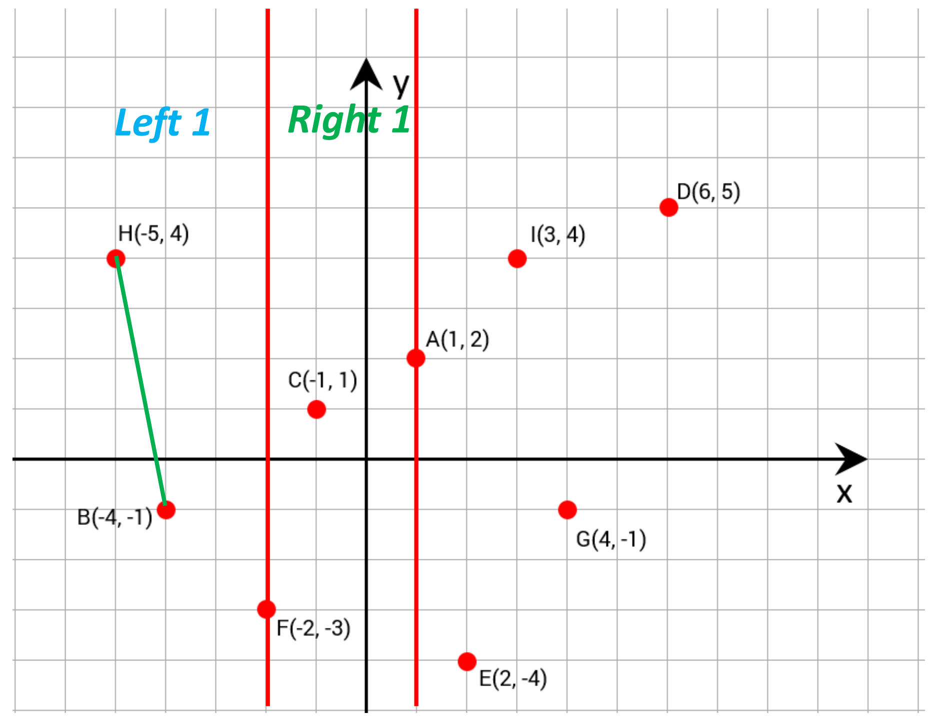
Bước 2.1: Tính khoảng cách các đoạn nhỏ

Left (0) Mid (2) Right (4)

Tên	H	B	F	C	A	E	I	G	D
Tọa độ	(-5, 4)	(-4, -1)	(-2, -3)	(-1, 1)	(1, 2)	(2, -4)	(3, 4)	(4, -1)	(6, 5)

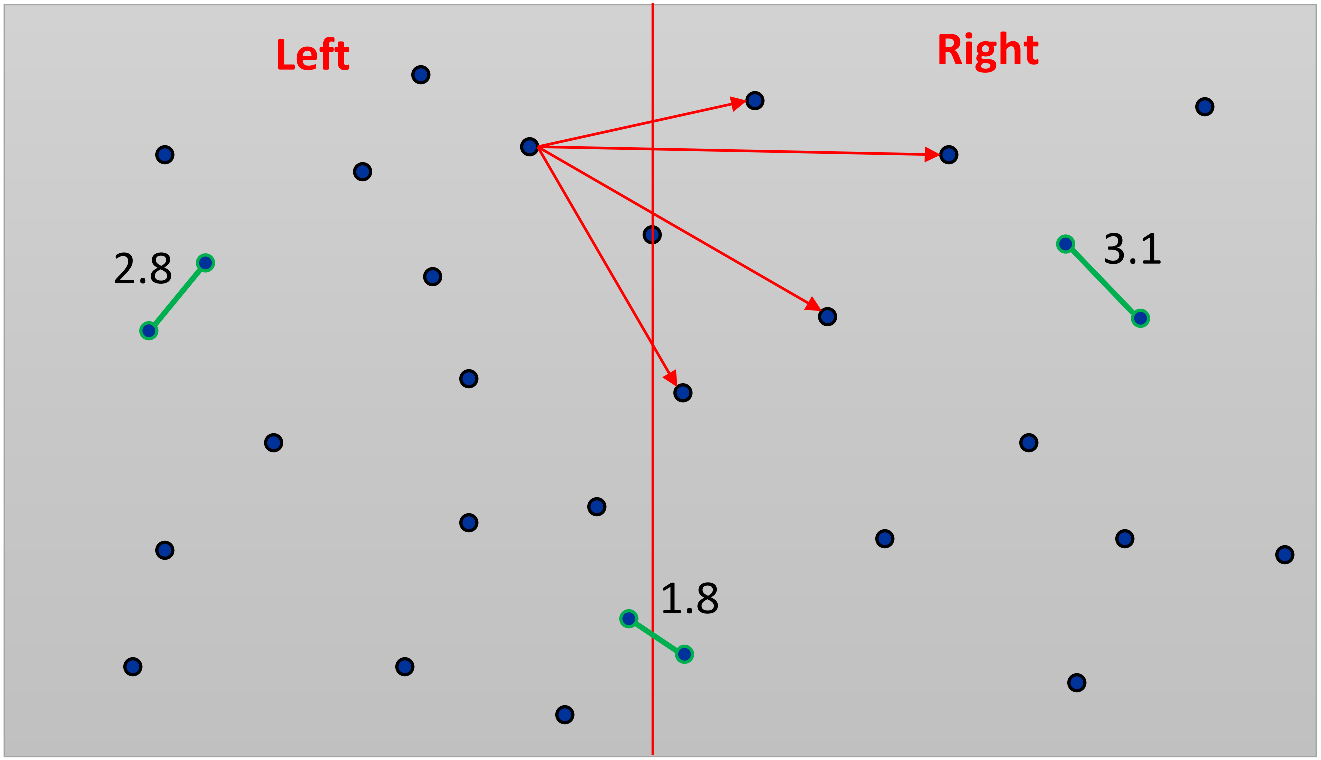
- $dist\_left\ (HB) = 5.09$
- $dist\_right\ (C) = \infty$ (Do bên Right chỉ có 1 điểm C nên khoảng cách giữa C và ∞ là ∞)

➔ $dist\_\min = 5.09$



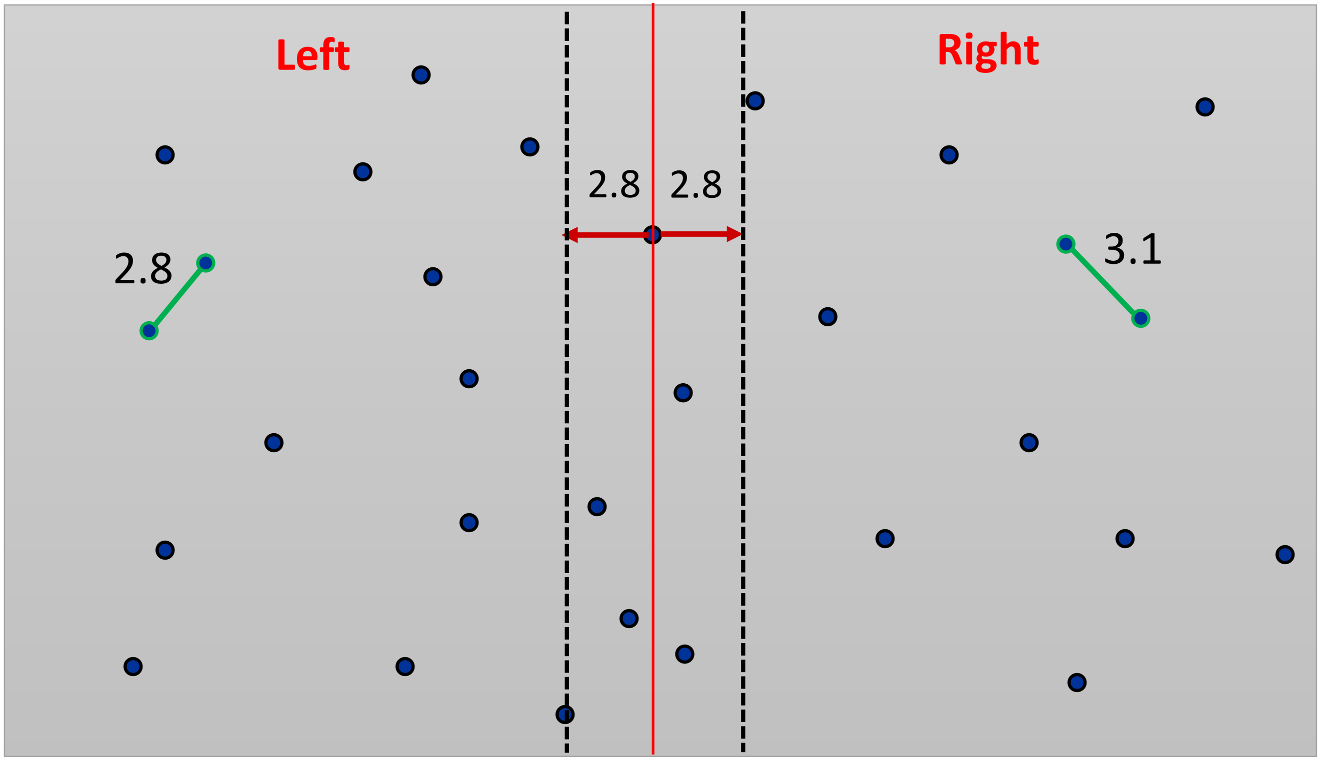
Phương pháp chia để trị

- Trường hợp khó khăn khi **Combine** là tính khoảng cách giữa các điểm thuộc 2 phía **Left, Right** có thể làm độ phức tạp của bài toán tăng lên rất lớn ➔ **Tìm giải pháp phù hợp.**



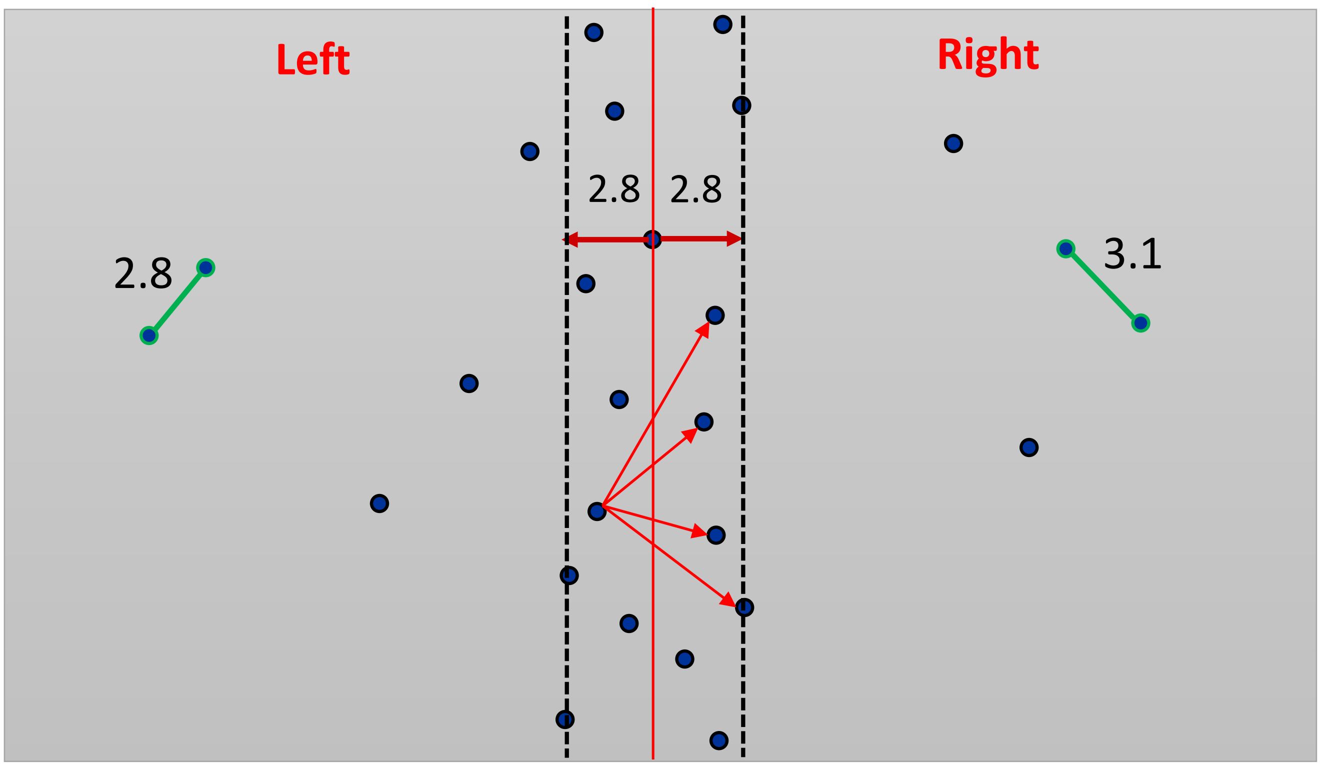
Phương pháp chia để trị

- Từ điểm Mid (điểm chia bài toán ra làm 2 phần) chỉ lấy những điểm thuộc khoảng cách **dist_min** đã tìm ở bước trước.
- Tìm đoạn nhỏ nhất trong khu vực này, nếu nhỏ hơn dist_min thì cập nhật lại dist_min.



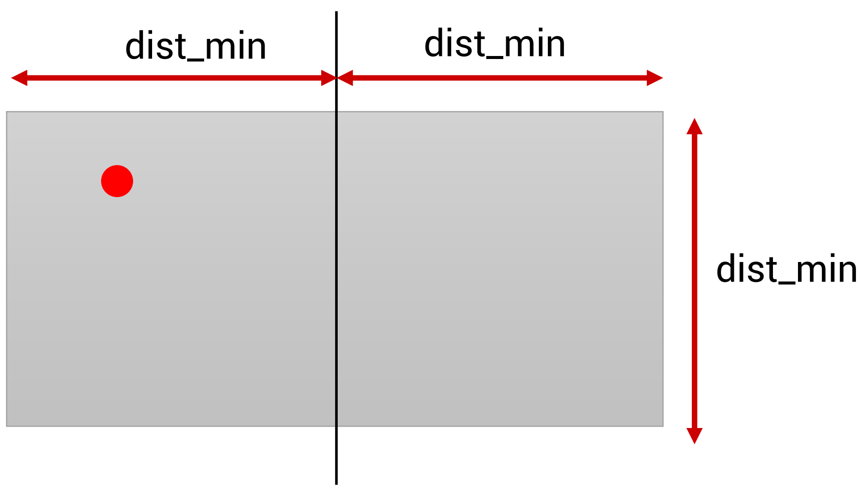
Phương pháp chia để trị

- **Warning:** Từ một điểm màu đỏ, phải tìm khoảng cách đến tất cả các điểm còn lại. Rồi tương tự từ 1 điểm khác cũng tìm khoảng cách đến tất cả các điểm còn lại ➔ **$O(N^2)$**



Phương pháp chia để trị

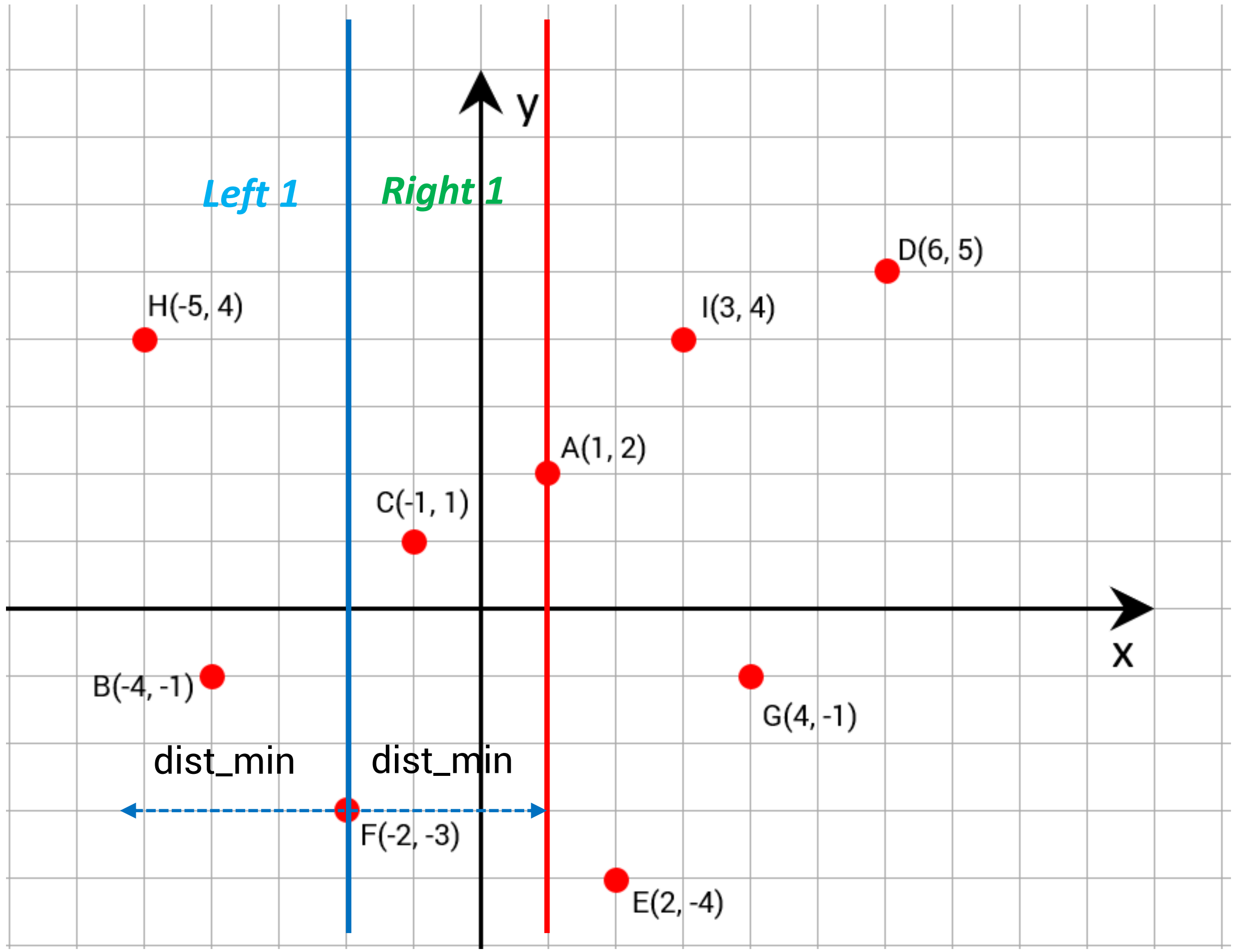
- **Giải pháp:** Sắp xếp các tọa độ theo **tung độ y**. Chọn những điểm **cần thiết** để tính khoảng cách (không chọn những điểm vượt quá dist_min).
- Lúc này mỗi điểm sẽ chỉ cần tìm đến không quá 6 điểm khác.



➔ Vẫn đảm bảo được được độ phức tạp **O(N)**

Bước 3: Tìm các điểm thuộc đoạn dist_min

Ta có dist_min = 5.09. Vì thế có 4 điểm sẽ nằm trong khoảng cách của dist_min: H, B, F, C.



Bước 4: Tìm khoảng cách ngắn nhất giữa 2 biên

Để giảm bớt việc xét nhiều khoảng cách vô ích, ta sắp xếp các tọa độ tăng dần theo tung độ y để tạo thành khung so sánh hình chữ nhật.

Tên	F	B	C	H
Tọa độ	(-2, -3)	(-4, -1)	(-1, 1)	(-5, 4)

Tính toán các độ dài, so sánh với dist_min (5.09):

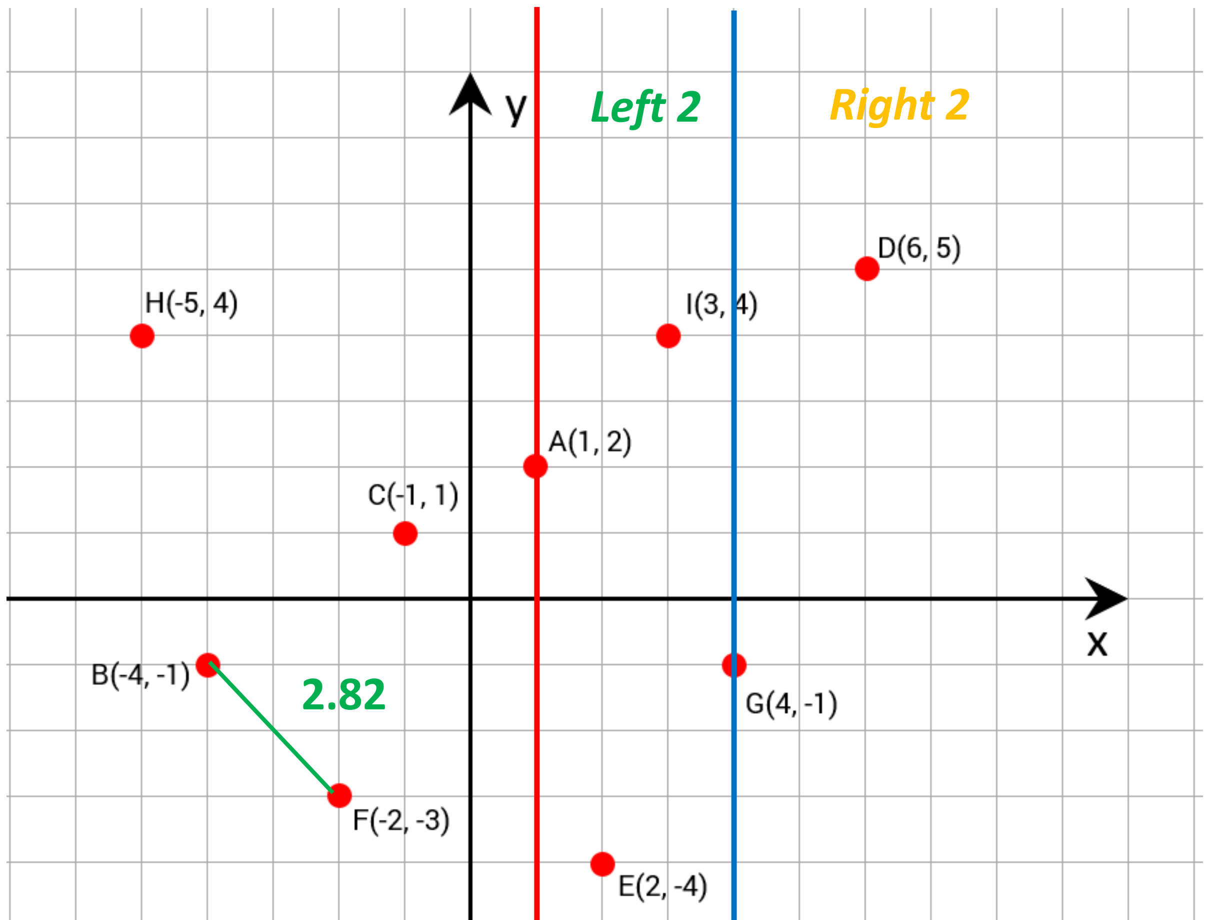
- FB = 2.82 ***(min)***
- FC = 4.12
- FH: (không cần tính, vì $y_H - y_F = 4 - (-3) = 7 > dist\_min$)
- BC = 3.60
- BH = 5.09
- CH = 5.00

$$\boldsymbol{dist}\_\min(\boldsymbol{FB}) = \mathbf{2.82}$$

Bước 2.2: chia mảng Right thành 2 phần

Chuyển qua phần bên Right ở bước 2 ban đầu, tính khoảng cách nhỏ nhất của phần bên này.

Tên	H	B	F	C	A	E	I	G	D
Tọa độ	(-5, 4)	(-4, -1)	(-2, -3)	(-1, 1)	(1, 2)	(2, -4)	(3, 4)	(4, -1)	(6, 5)



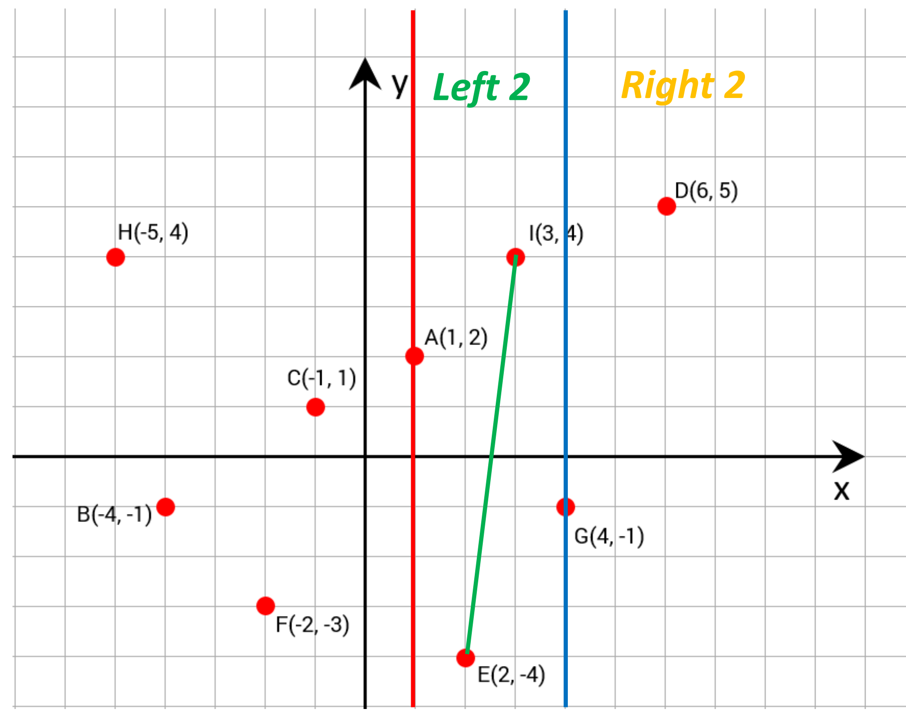
Bước 2.2: Tính khoảng cách các đoạn nhỏ

Left | Mid | Right

Tên	H	B	F	C	A	E	I	G	D
Tọa độ	(-5, 4)	(-4, -1)	(-2, -3)	(-1, 1)	(1, 2)	(2, -4)	(3, 4)	(4, -1)	(6, 5)

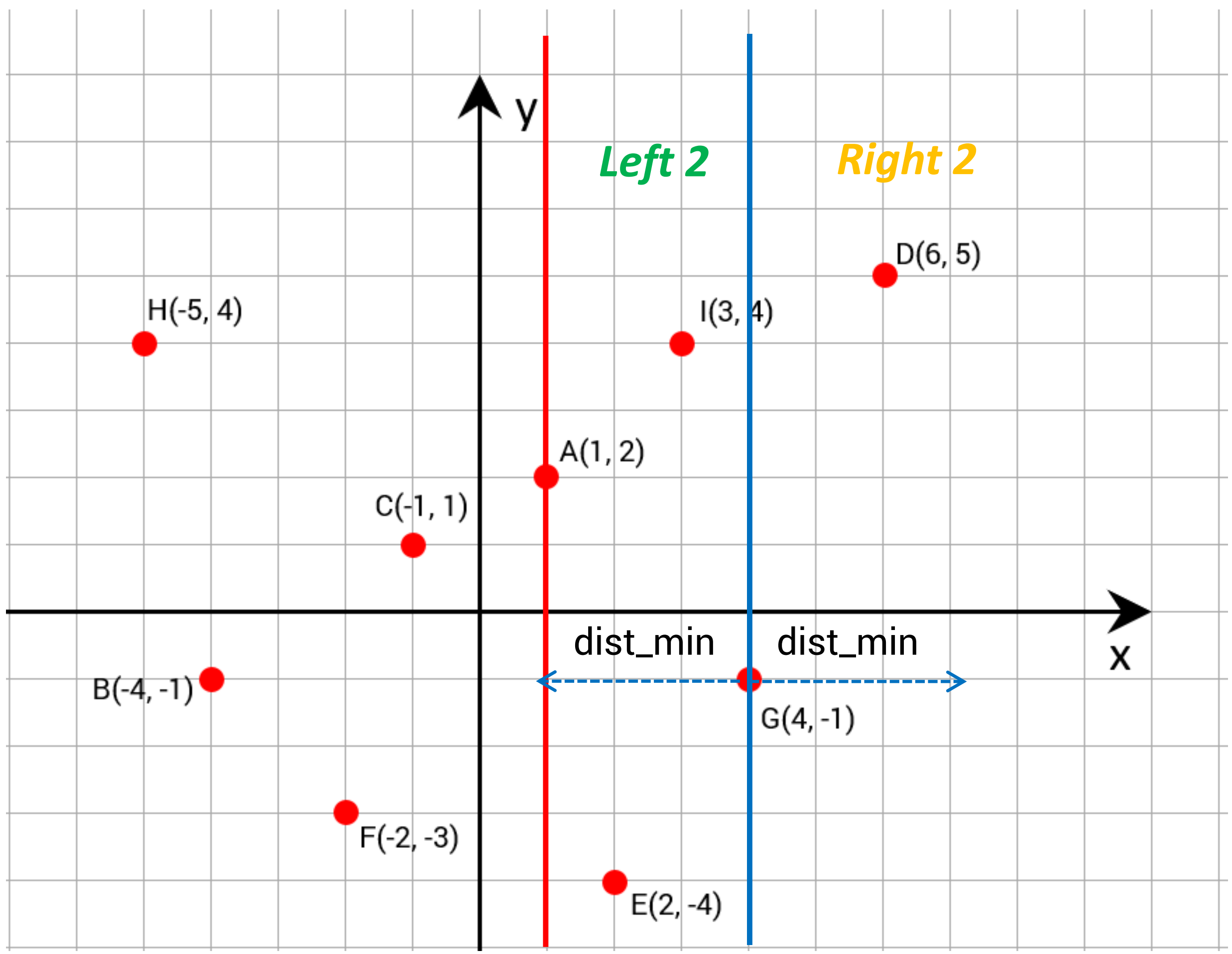
- $dist\_left\ (EI) = 8.06$
- $dist\_right\ (D) = \infty$ (Do bên Right chỉ có 1 điểm D nên khoảng cách giữa D và ∞ là ∞)

➔ $dist\_min = 8.06$



Bước 3: Tìm các điểm thuộc đoạn dist_min

Ta có $dist\_min = 8.06$. Vì thế có 4 điểm sẽ nằm trong khoảng cách của dist_min: E, I, G, D.



Bước 4: Tìm khoảng cách ngắn nhất giữa 2 biên

Để giảm bớt việc xét nhiều khoảng cách vô ích, ta sắp xếp các tọa độ tăng dần theo tung độ y để tạo thành khung so sánh hình chữ nhật.

Tên	E	G	I	D
Tọa độ	(2, -4)	(4, -1)	(3, 4)	(6, 5)

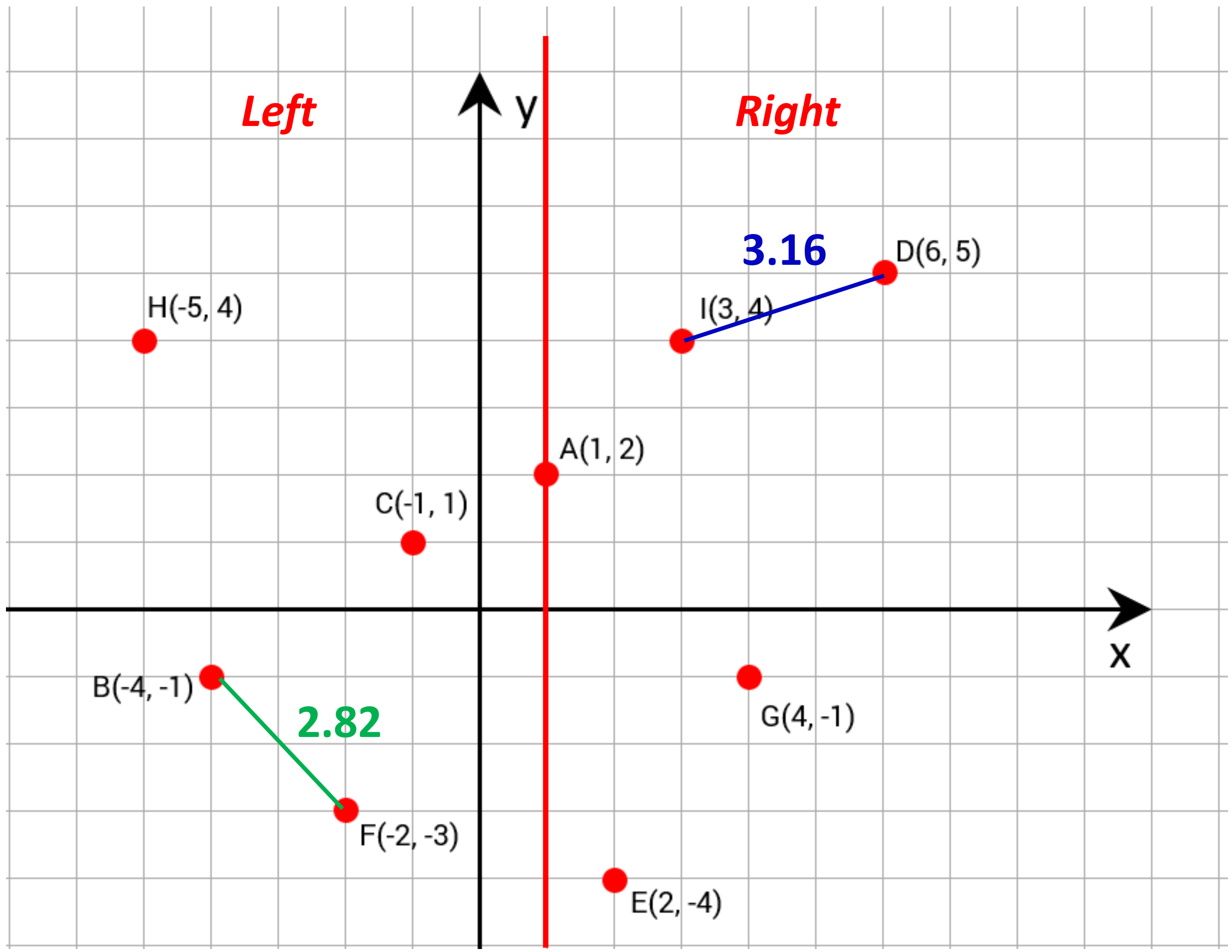
Tính toán các độ dài, so sánh với dist_min (8.06):

- EG = 3.60
- EI = 8.06
- ED: (không cần tính, vì $y_D - y_E = 5 - (-4) = 9 > dist\_min$)
- GI = 5.09
- GD = 6.32
- ID = 3.16 ***(min)***

$$\boldsymbol{dist}\_\mathrm{min}(\boldsymbol{ID}) = \mathbf{3.16}$$

Tóm tắt lại sau 2 quá trình phân chia

- Đoạn ngắn nhất bên nữa Left: FB = 2.82
- Đoạn ngắn nhất bên nữa Right: ID = 3.16



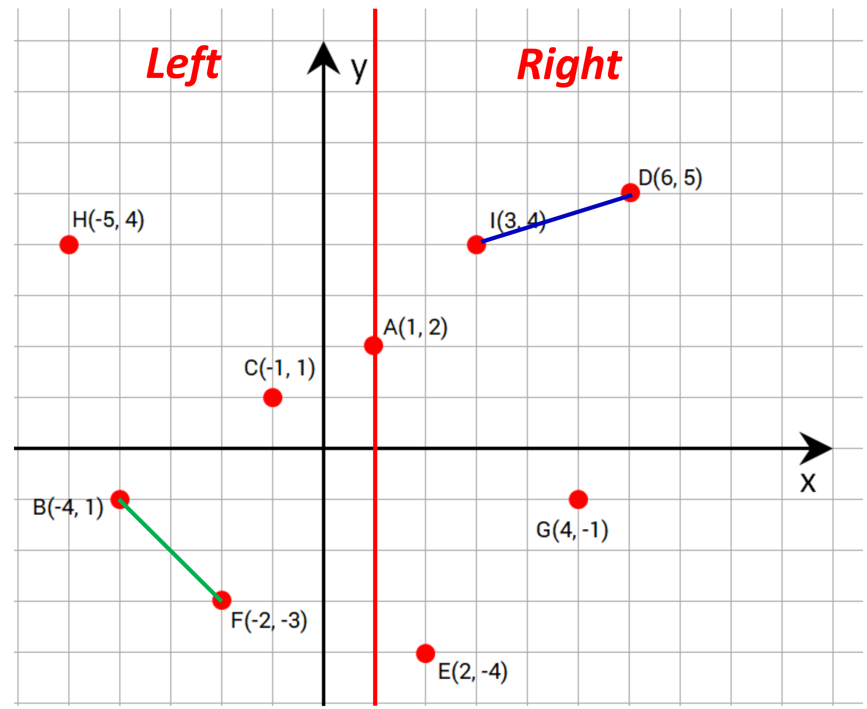
Chuyển lên bước 2: Tính khoảng cách các đoạn nhỏ

Left (0) Mid (4) Right (0)

Tên	H	B	F	C	A	E	I	G	D
Tọa độ	(-5, 4)	(-4, -1)	(-2, -3)	(-1, 1)	(1, 2)	(2, -4)	(3, 4)	(4, -1)	(6, 5)

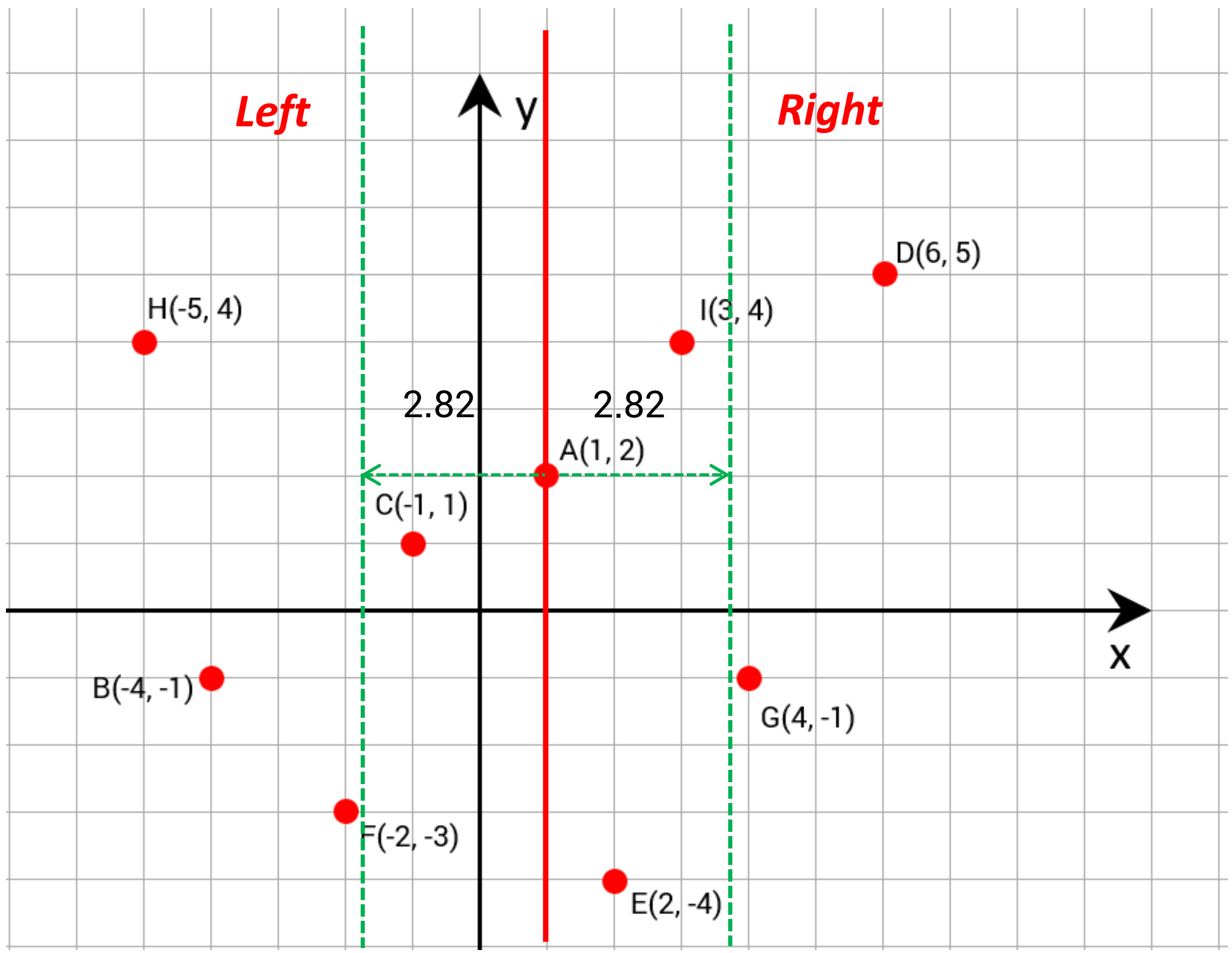
- $dist\_left\ (BF)\ =\ 2.82$
- $dist\_right\ (ID)\ =\ 3.16$

➔ $dist\_\min =\ 2.82$



Bước 3: Tìm các điểm thuộc đoạn dist_min

Ta có $dist\_\min = \ 2.82$. Vì thế có 4 điểm sẽ nằm trong khoảng cách của dist_min: C, A, E, I.



Bước 4: Tìm khoảng cách ngắn nhất giữa 2 biên

Sắp xếp các tọa độ tăng dần theo Oy để tạo thành khung so sánh hình chữ nhật.

Tên	E	C	A	I
Tọa độ	(2, -4)	(-1, 1)	(1, 2)	(3, 4)

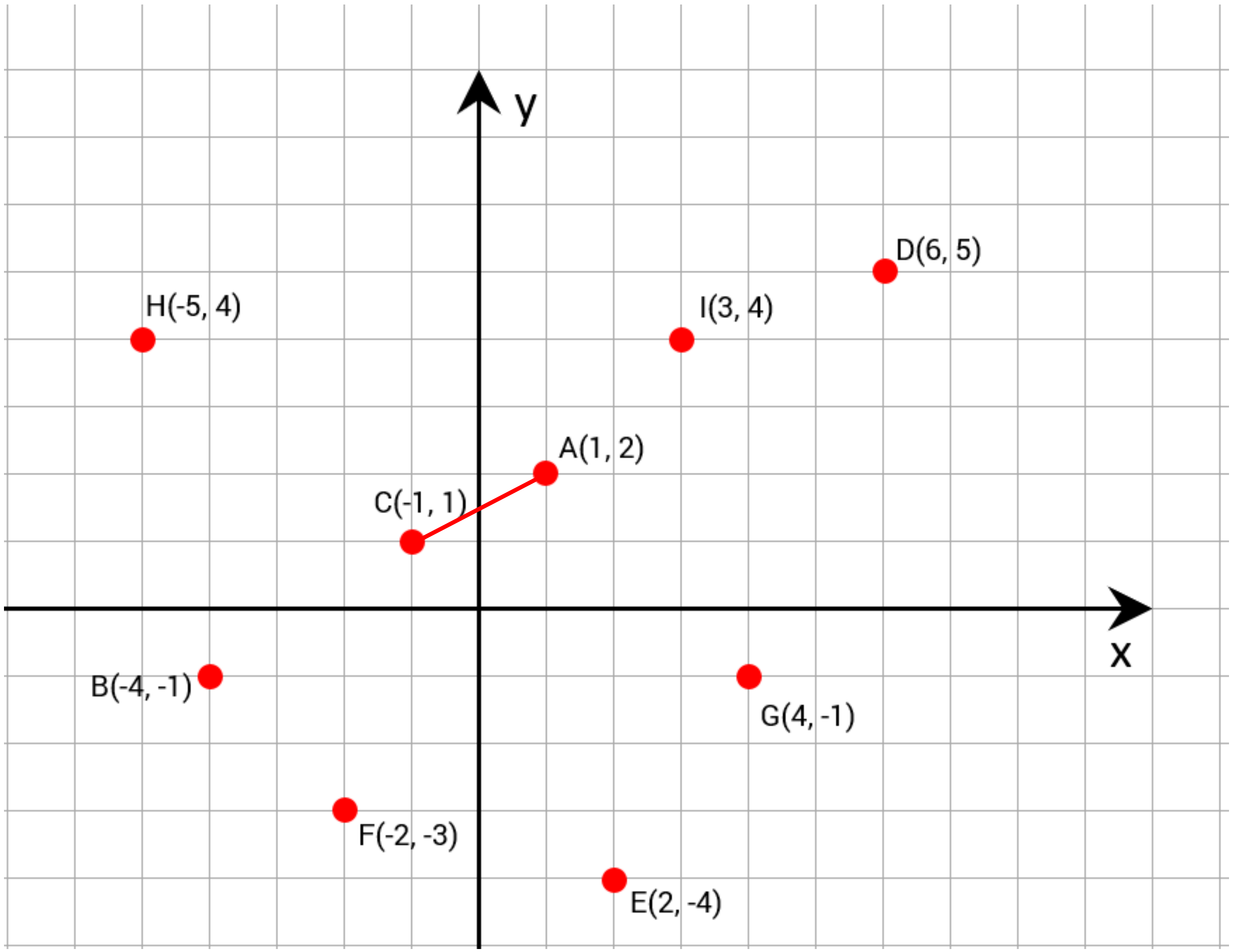
Tính toán các độ dài, so sánh với dist_min (2.82):

- EC: bỏ qua vì $y_C - y_E = 1 - (-4) = 5 > dist\_min$
- EA: bỏ qua vì $y_A - y_E = 2 - (-4) = 6 > dist\_min$
- EI: bỏ qua vì $y_I - y_E = 4 - (-4) = 8 > dist\_min$
- CA = 2.24 ***(min)***
- CI: bỏ qua vì $y_I - y_C = 4 - 1 = 3 > dist\_min$
- AI = 2.82

$\boldsymbol{dist}\_\mathrm{min}(\boldsymbol{CA}) = \mathbf{2.24}$

Kết quả bài toán

Cặp điểm gần nhất là CA = 2.24



Tóm tắt lại phương pháp cải tiến

Bước 1: Sắp xếp tọa độ điểm theo thứ tự tăng dần của hoành độ (Ox).

Bước 2: Chia mảng ra làm 2 phần (Left và Right). Tìm khoảng cách ngắn nhất mỗi bên. So sánh bên nào ngắn hơn chọn khoảng cách đó. Gọi là $dist\_\mathrm{min}$.

Bước 3: Trường hợp khoảng cách ngắn nhất nằm 2 điểm thuộc nửa này và nửa kia. Dùng $dist\_\mathrm{min}$ để chọn 2 điểm nằm khác phía nhau.

Bước 4: Tránh việc so sánh nhiều lần các điểm trong tập vừa tìm được, sắp xếp các điểm theo tung độ (Oy). Tạo thành khung so sánh hình chữ nhật. Tính khoảng cách các điểm và cập nhật $dist\_\mathrm{min}$.

Bước 5: Xuất kết quả cần tìm.

Độ phức tạp: **O(NlogN)**

Source Code Closest Pair of Points

```cpp
#include <iostream>
#include <cmath>
#include <vector>
#include <algorithm>
#include <iomanip>
using namespace std;
#define INF 1e9
struct Point
{
    double x, y;
};
bool xCompare(const Point &p1, const Point &p2)
{
    return p1.x < p2.x;
}
bool yCompare(const Point &p1, const Point &p2)
{
    return p1.y < p2.y;
}
```

Source Code Closest Pair of Points

```cpp
double distance(Point &p1, Point &p2)
{
    double x = p1.x - p2.x;
    double y = p1.y - p2.y;
    return sqrt(x * x + y * y);
}

double bruteForce(vector<Point> &point_set, int left, int right)
{
    double min_dist = INF;
    for (int i = left; i < right; ++i)
        for (int j = i + 1; j < right; ++j)
            min_dist = min(min_dist, distance(point_set[i], point_set[j]));
    return min_dist;
}
```

Source Code Closest Pair of Points

```cpp
double stripClosest(vector<Point> &strip, double dist_min)
{
    sort(strip.begin(), strip.end(), yCompare);
    double min = dist_min;
    for (int i = 0; i < strip.size(); i++)
        for (int j = i + 1; j < strip.size() && (strip[j].y - strip[i].y) < min; j++)
            if (distance(strip[i], strip[j]) < min)
                min = distance(strip[i], strip[j]);
    return min;
}
```

Source Code Closest Pair of Points

```cpp
double minimalDistance(vector<Point> &xPoints, vector<Point> &yPoints, int left, int right)
{
    if (right - left <= 3)
        return bruteForce(xPoints, left, right);
    int mid = (right + left) / 2;
    Point midPoint = xPoints[mid];
    double dist_left = minimalDistance(xPoints, yPoints, left, mid);
    double dist_right = minimalDistance(xPoints, yPoints, mid + 1, right);
    double dist_min = min(dist_left, dist_right);
    vector<Point> strip;
    for (int i = left; i < right; i++)
        if (abs(xPoints[i].x - midPoint.x) < dist_min)
            strip.push_back(xPoints[i]);
    return min(dist_min, stripClosest(strip, dist_min));
}
```

Source Code Closest Pair of Points

```
int main()
{
    int n;
    Point p;
    cin >> n;
    vector<Point> xPoints, yPoints;
    for (int i = 0; i < n; i++)
    {
        cin >> p.x >> p.y;
        xPoints.push_back(p);
        yPoints.push_back(p);  //có thể bỏ
    }
    sort(xPoints.begin(), xPoints.end(), xCompare);
    double result = minimalDistance(xPoints, yPoints,  0, n);
    cout << fixed << setprecision(2) << result << endl;
    return 0;
}
```

Source Code Closest Pair of Points

```python
import math
INF = 1e9
class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
def distance(p1, p2):
    x = p1.x - p2.x
    y = p1.y - p2.y
    return math.sqrt(x * x + y * y)
def bruteForce(points, left, right):
    min_dist = INF
    for i in range(left, right):
        for j in range(i + 1, right):
            min_dist = min(min_dist, distance(points[i], points[j]))
    return min_dist
```

Source Code Closest Pair of Points

```python
def stripClosest(strip, dist_min):
    strip.sort(key = lambda p: p.y)
    min = dist_min
    for i in range(len(strip)):
        for j in range(i + 1, len(strip)):
            if strip[j].y - strip[i].y >= min:
                break
            if distance(strip[i], strip[j]) < min:
                min = distance(strip[i], strip[j])
    return min
```

Source Code Closest Pair of Points

```python
def minimalDistance(xPoints, yPoints, left, right):
    if right - left <= 3:
        return bruteForce(xPoints, left, right)
    mid = (right + left) // 2
    midPoint = xPoints[mid]
    dist_left = minimalDistance(xPoints, yPoints, left, mid)
    dist_right = minimalDistance(xPoints, yPoints, mid + 1, right)
    dist_min = min(dist_left, dist_right)
    strip = []
    for i in range(left, right):
        if abs(xPoints[i].x - midPoint.x) < dist_min:
            strip.append(xPoints[i])
    return min(dist_min, stripClosest(strip, dist_min))
```

Source Code Closest Pair of Points

```python
if __name__ == "__main__":
    n = int(input())
    xPoints = []
    yPoints = []
    for i in range(n):
        x, y = map(float, input().split())
        p = Point(x, y)
        xPoints.append(p)
        yPoints.append(p)
    xPoints.sort(key = lambda p: p.x)
    yPoints.sort(key = lambda p: p.y)
    result = minimalDistance(xPoints, yPoints, 0, n)
    print('{:.2f}'.format(result))
```

Big-O Orange

Source Code Closest Pair of Points

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;
class Point {
    public Double x, y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
public class Main {
    private static final int INF = (int)1e9;

    private static double distance(Point p1, Point p2) {
        double x = p1.x - p2.x;
        double y = p1.y - p2.y;
        return Math.sqrt(x*x + y*y);
    }
```

Source Code Closest Pair of Points

```java
    private static double bruteForce(Point[] points, int left, int right) {
        double min_dist = INF;
        for (int i = left; i < right; ++i)
            for (int j = i + 1; j < right; ++j)
                min_dist = Math.min(min_dist, distance(points[i], points[j]));
        return min_dist;
    }
    private static double stripClosest(ArrayList<Point> strip, double dist_min) {
        Arrays.sort(strip, (o1, o2) -> o1.y.compareTo(o2.y));
        double min = dist_min;
        for (int i = 0; i < strip.size(); i++)
            for (int j = i + 1; j < strip.size() && (strip.get(j).y -
                                      strip.get(i).y) < min; j++)
                if (distance(strip.get(i), strip.get(j)) < min)
                    min = distance(strip.get(i), strip.get(j));
        return min;
    }
```

Source Code Closest Pair of Points

```java
    private static double minimalDistance(Point[] xPoints, Point[] yPoints, int left, int right) {
        if (right - left <= 3)
            return bruteForce(xPoints, left, right);
        int mid = (right + left) / 2;
        Point midPoint = xPoints[mid];
        double dist_left = minimalDistance(xPoints, yPoints, left, mid);
        double dist_right = minimalDistance(xPoints, yPoints, mid + 1, right);
        double dist_min = Math.min(dist_left, dist_right);
        ArrayList<Point> strip = new ArrayList<>();
        for (int i = left; i < right; i++)
            if (Math.abs(xPoints[i].x - midPoint.x) < dist_min)
                strip.add(xPoints[i]);
        return Math.min(dist_min, stripClosest(strip, dist_min));
    }
```

Source Code Closest Pair of Points

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        double x, y;
        Point[] xPoints = new Point[n];
        Point[] yPoints = new Point[n];
        for (int i = 0; i < n; i++) {
            x = sc.nextDouble();
            y = sc.nextDouble();
            xPoints[i] = new Point(x, y);
            yPoints[i] = new Point(x, y);  //có thể bỏ
        }
        Arrays.sort(xPoints, (o1, o2) -> o1.x.compareTo(o2.x));
        Arrays.sort(yPoints, (o1, o2) -> o1.y.compareTo(o2.y));
        double result = minimalDistance(xPoints, yPoints,  0, n);
        System.out.printf("%.2f\n", result);
    }
}
```

Nhận xét sau bài toán 1

Divide and Conquer tồn tại 2 khó khăn:

- Làm thế nào để chia tách bài toán một cách hợp lý thành các bài toán con, bởi vì nếu các bài toán con được giải quyết bằng các thuật toán khác nhau thì sẽ rất phức tạp.
- Việc kết hợp lời giải các bài toán con lại với nhau để tìm kết quả bài toán lớn ban đầu được thực hiện như thế nào?

Bài toán minh họa 2

Maximum Subarray Sum hay tên gọi khác **Largest Sum Contiguous Subarray:** Cho mảng một chiều có cả số âm và số dương hãy tìm mảng con có tổng lớn nhất.

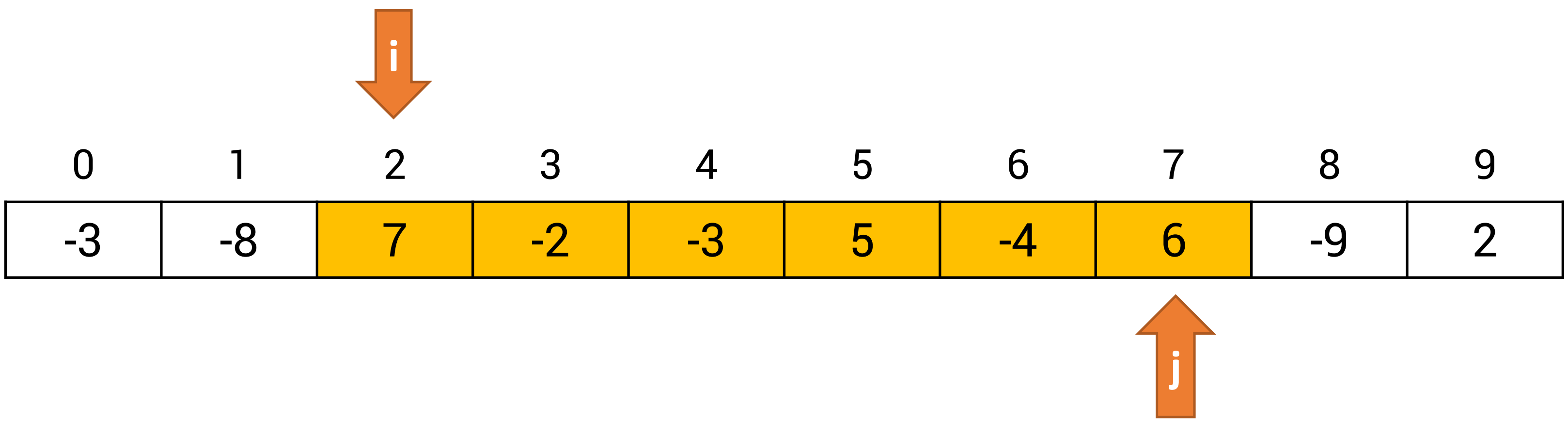
0	1	2	3	4	5	6	7	8	9
-3	-8	7	-2	-3	5	-4	6	-9	2

Tổng mảng con: 7 + (-2) + (-3) + 5 + (-4) + 6 = 9

Phương pháp giải quyết thông thường

Dùng 2 vòng lặp for lồng nhau để tìm kiếm đoạn chứa tổng giá trị là lớn nhất.

- Biến i: vị trí bắt đầu của mảng con.
- Biến j: vị trí kết thúc của mảng con.



Độ phức tạp: $O(N^2)$

Phương pháp giải quyết thông thường

maxSum = a[0]

i

0	1	2	3	4	5	6	7	8	9
-3	-8	7	-2	-3	5	-4	6	-9	2

j

i

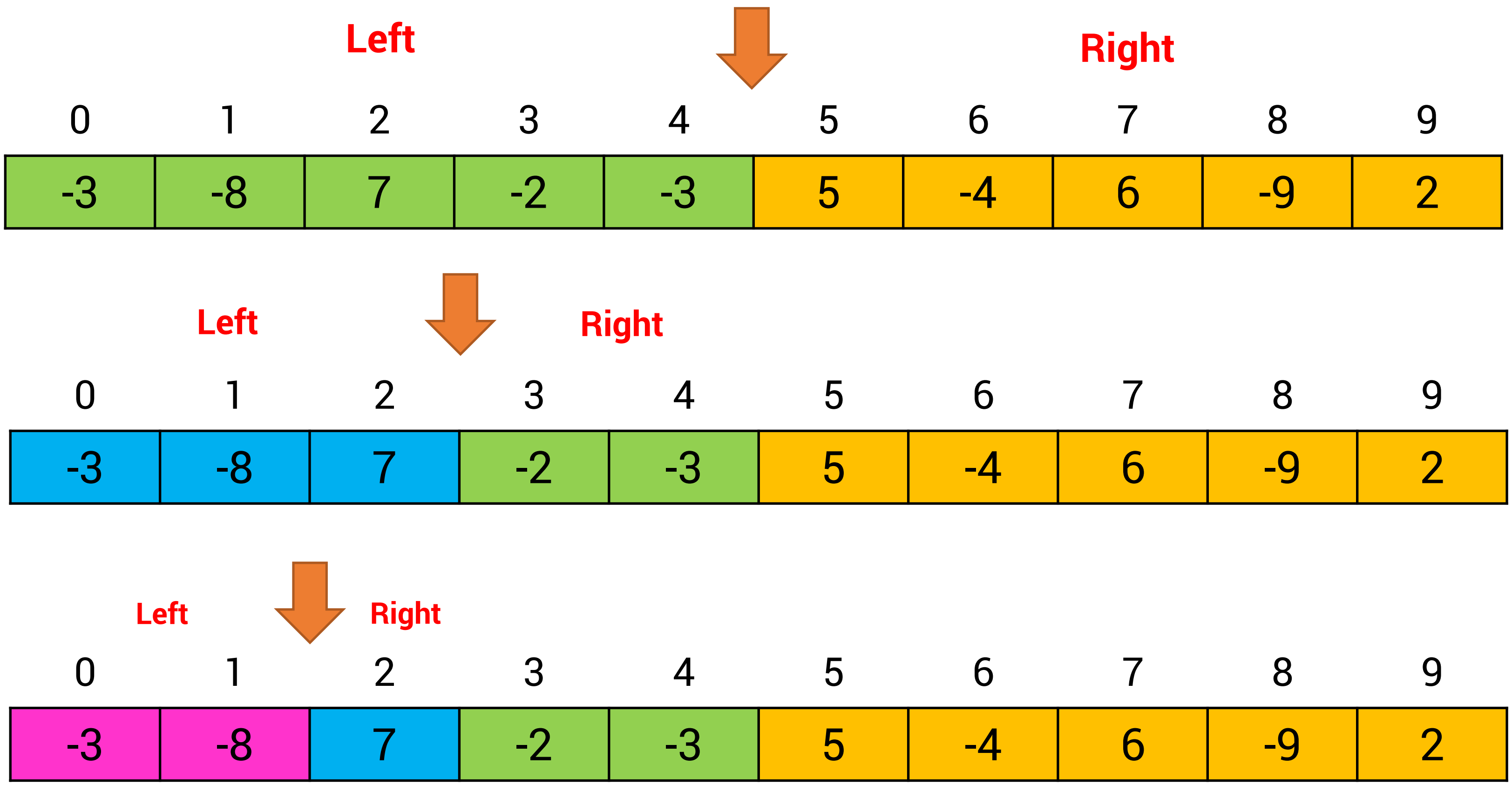
0	1	2	3	4	5	6	7	8	9
-3	-8	7	-2	-3	5	-4	6	-9	2

j

maxSum = 9

Phương pháp chia để trị

- **Divide: Chia** bài toán lớn thành các bài toán con nhỏ hơn.



Phương pháp chia để trị

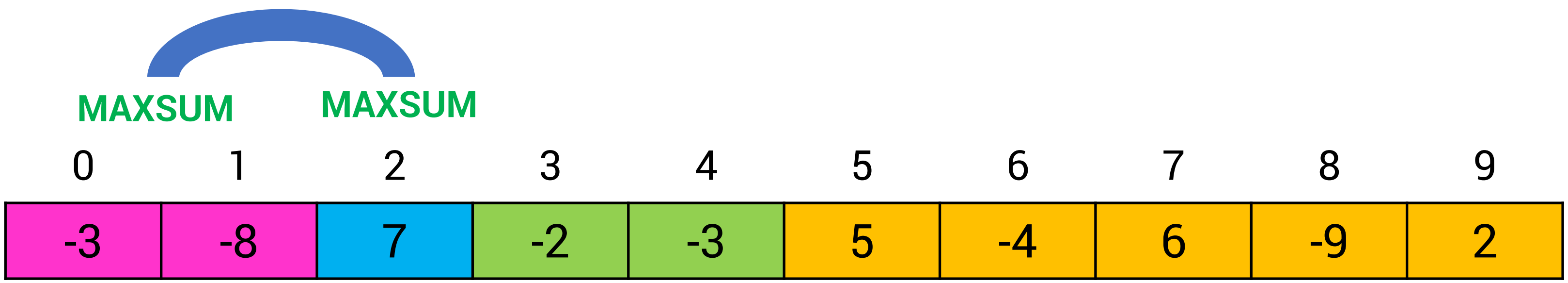
- **Divide:** **Chia** bài toán lớn thành các bài toán con nhỏ hơn.
- **Conquer:** Sử dụng tính chất đệ quy để **trị** (giải) các bài toán con.

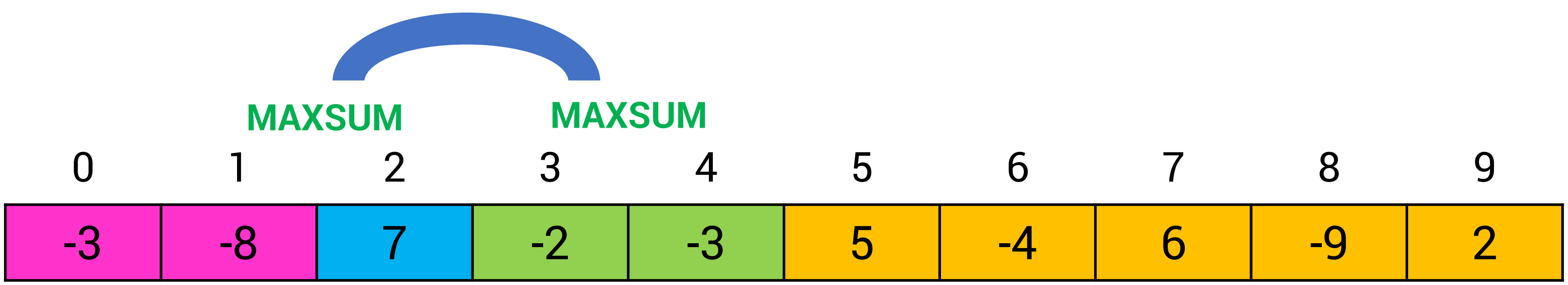
MAXSUM

0	1	2	3	4	5	6	7	8	9
-3	-8	7	-2	-3	5	-4	6	-9	2

Phương pháp chia để trị

- **Divide:** **Chia** bài toán lớn thành các bài toán con nhỏ hơn.
- **Conquer:** Sử dụng tính chất đệ quy để **trị** (giải) các bài toán con.
- **Combine:** **Ghép** lời giải của các bài toán con, để giải bài toán lớn ban đầu.





Phương pháp chia để trị

- **Divide: Chia** bài toán lớn thành các bài toán con nhỏ hơn.
- **Conquer:** Sử dụng tính chất đệ quy để **trị** (giải) các bài toán con.
- **Combine: Ghép** lời giải của các bài toán con, để giải bài toán lớn ban đầu.

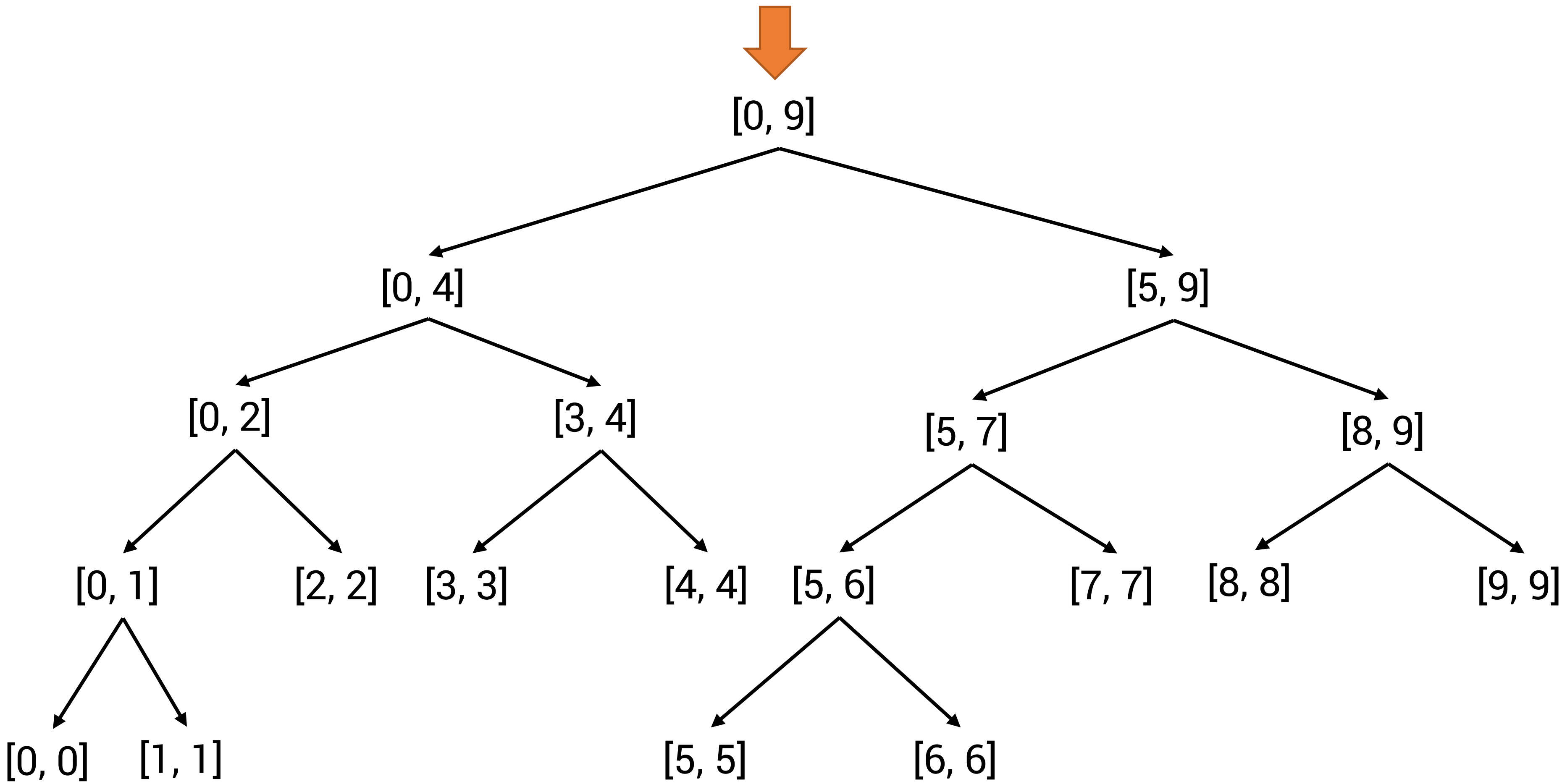
MAXSUM

0	1	2	3	4	5	6	7	8	9
-3	-8	7	-2	-3	5	-4	6	-9	2

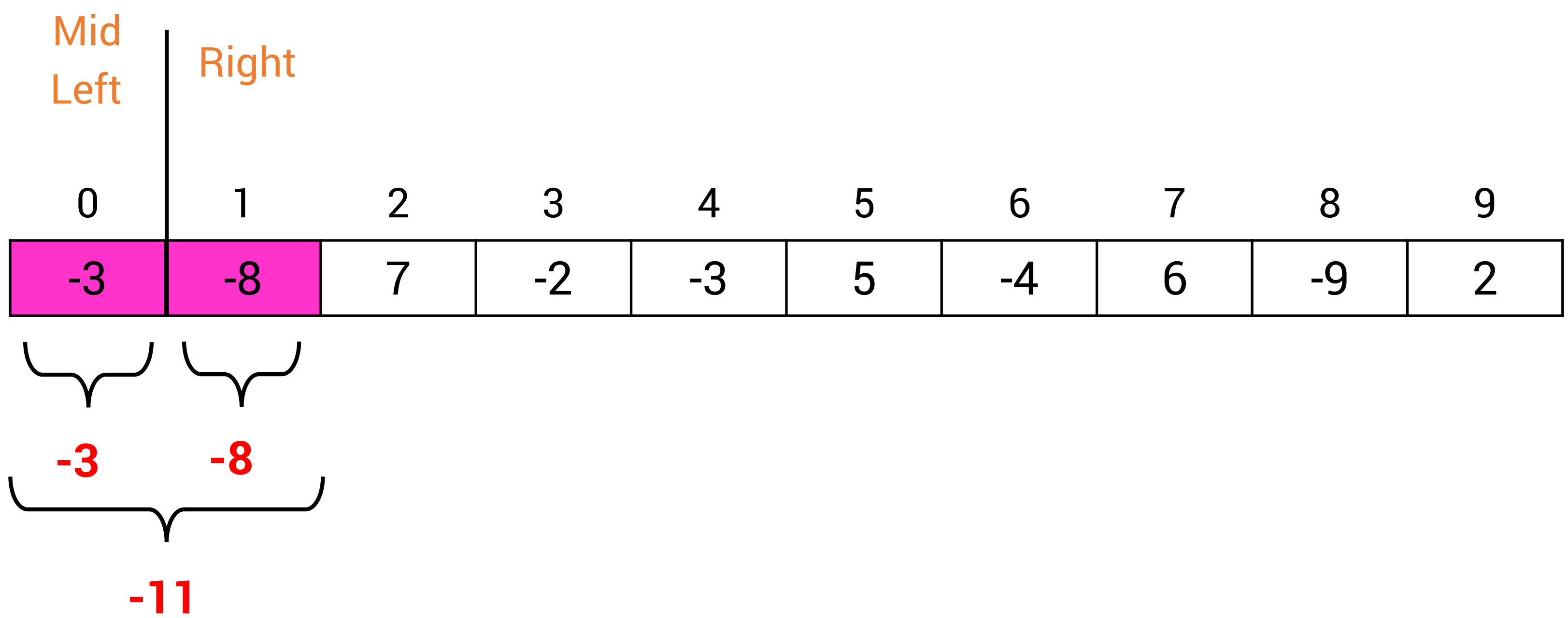
Độ phức tạp: O(NlogN)

Bước 1: Chia mảng ra thành 2 phần

Lần lượt chia mảng ban đầu thành 2 phần, chia cho đến khi gặp trường hợp nhỏ nhất có thể tìm mảng con có tổng lớn nhất một cách dễ dàng.



Bước 2: Tính tổng mảng đoạn [0, 1]

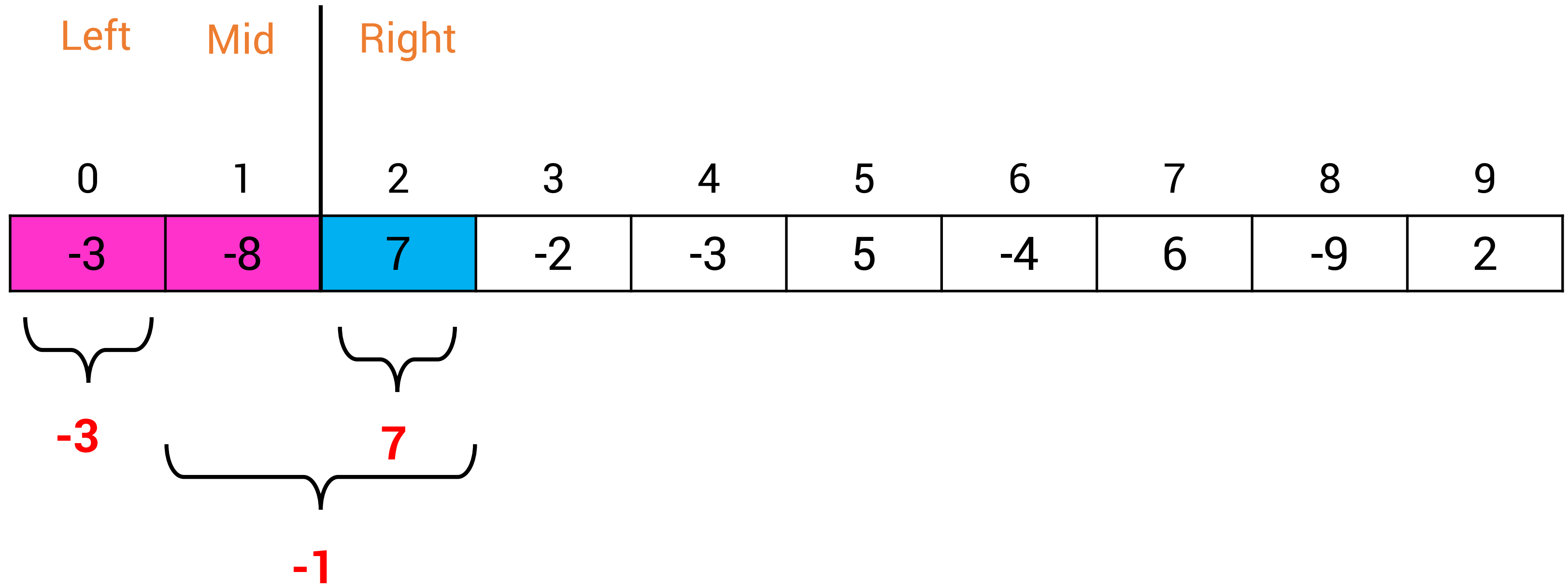


- maxSum_left: Tổng mảng lớn nhất bên Left [0, 0].
- maxSum_right: Tổng mảng lớn nhất bên Right [1, 1].
- maxSum_mid: Tổng mảng lớn nhất ghép cả Left và Right lại [0, 1].

maxSum_left	maxSum_right	maxSum_mid
-3	-8	-11

result = **max**(-3, -8, -11) = -3

Bước 3: Tính tổng mảng đoạn [0, 2]

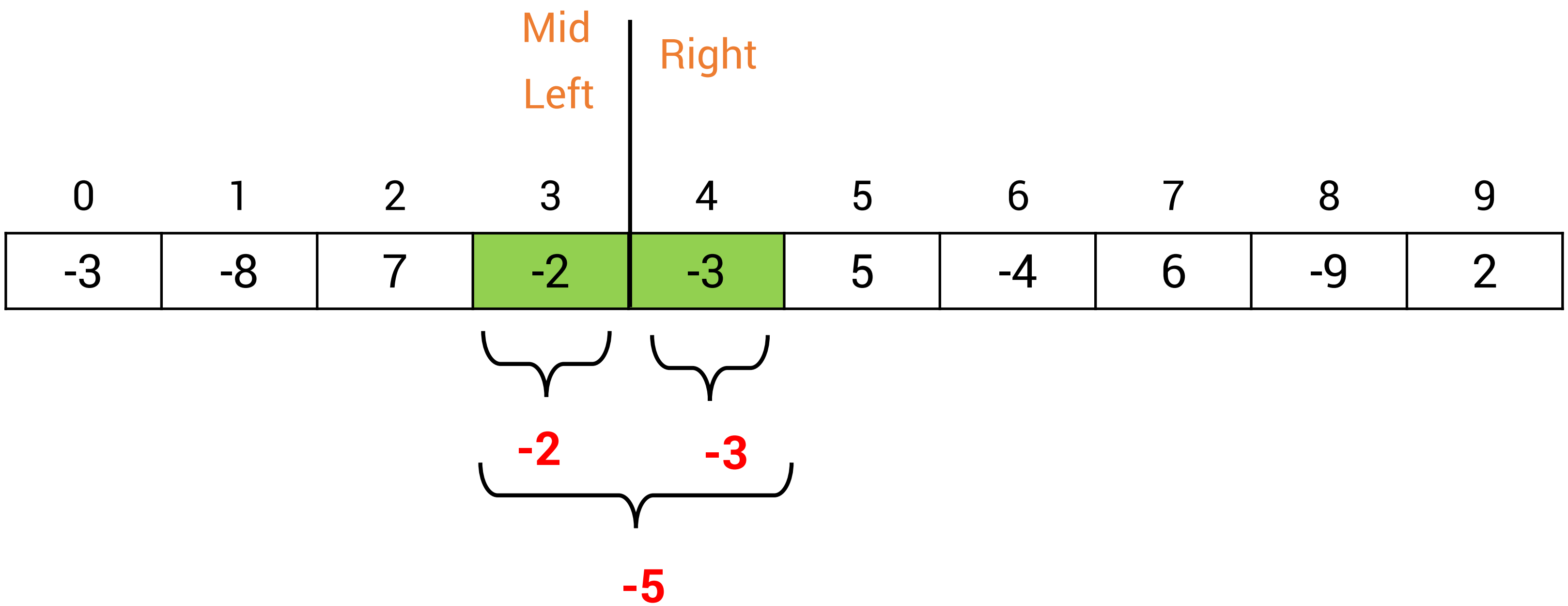


- maxSum_left: Tổng mảng lớn nhất bên Left [0, 1].
- maxSum_right: Tổng mảng lớn nhất bên Right [2, 2].
- maxSum_mid: Tổng mảng lớn nhất ghép cả Left và Right lại [1, 2].

maxSum_left	maxSum_right	maxSum_mid
-3	7	-1

result = **max**(-3, 7, -1) = 7

Bước 4: Tính tổng mảng đoạn [3, 4]

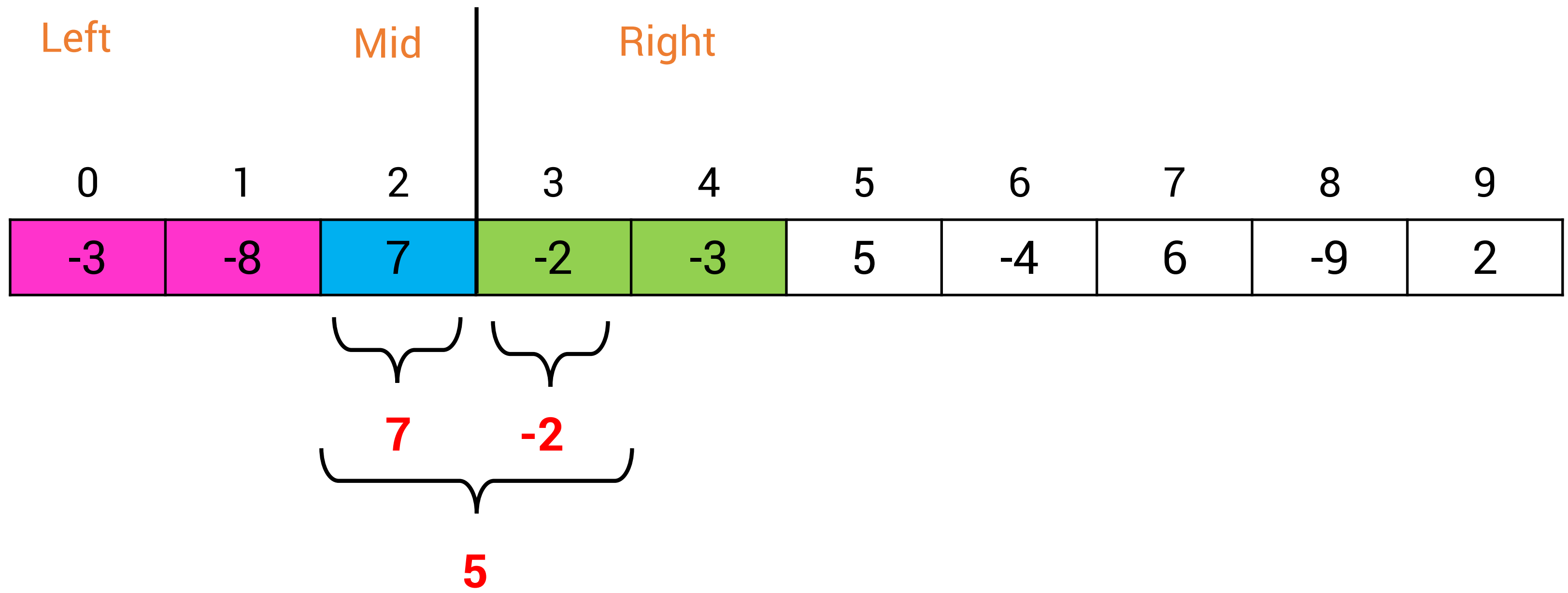


- maxSum_left: Tổng mảng lớn nhất bên Left [3, 3].
- maxSum_right: Tổng mảng lớn nhất bên Right [4, 4].
- maxSum_mid: Tổng mảng lớn nhất ghép cả Left và Right lại [3, 4].

maxSum_left	maxSum_right	maxSum_mid
-2	-3	-5

result = **max**(-2, -3, -5) = -2

Bước 5: Tính tổng mảng đoạn [0, 4]

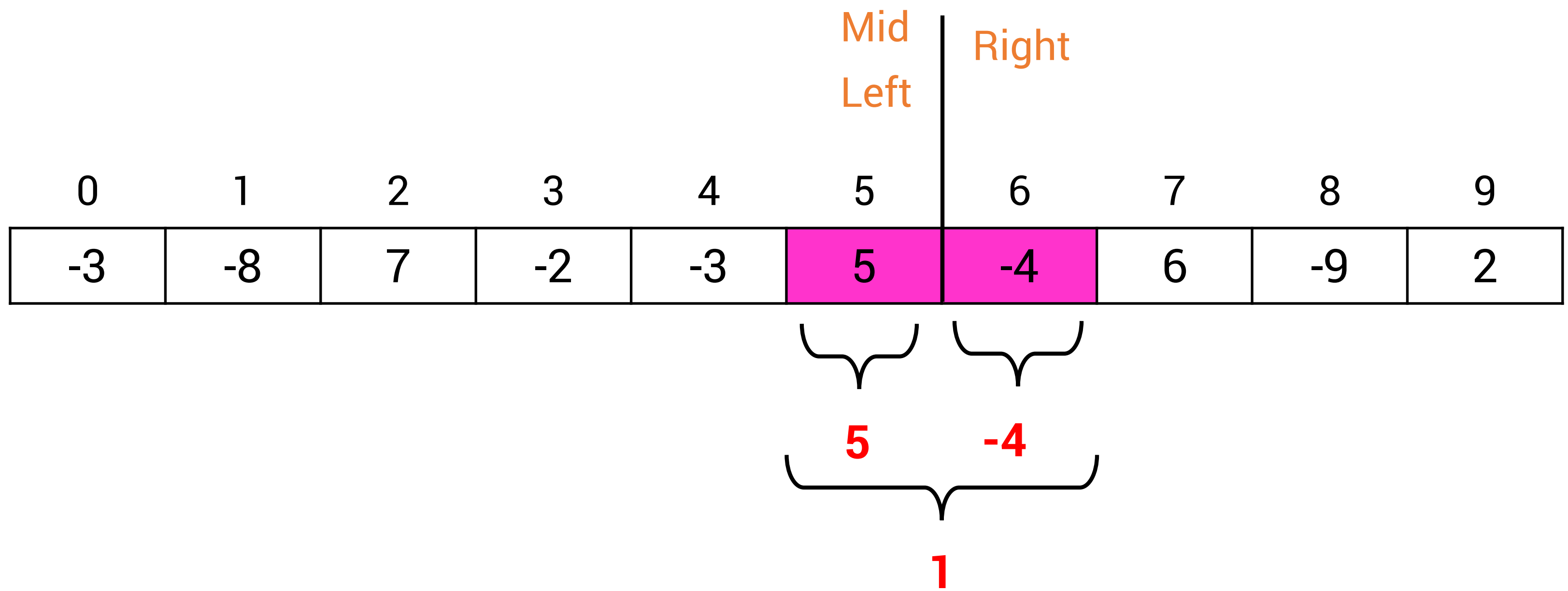


- maxSum_left: Tổng mảng lớn nhất bên Left [0, 2].
- maxSum_right: Tổng mảng lớn nhất bên Right [3, 4].
- maxSum_mid: Tổng mảng lớn nhất ghép cả Left và Right lại [2, 3].

maxSum_left	maxSum_right	maxSum_mid
7	-2	5

result = **max**(7, -2, 5) = 7

Bước 6: Tính tổng mảng đoạn [5, 6]



- maxSum_left: Tổng mảng lớn nhất bên Left [5, 5].
- maxSum_right: Tổng mảng lớn nhất bên Right [6, 6].
- maxSum_mid: Tổng mảng lớn nhất ghép cả Left và Right lại [5, 6].

maxSum_left	maxSum_right	maxSum_mid
5	-4	1

result = **max**(5, -4, 1) = 5

Bước 7: Tính tổng mảng đoạn [5, 7]

Left Mid Right

0	1	2	3	4	5	6	7	8	9
-3	-8	7	-2	-3	5	-4	6	-9	2

5 6

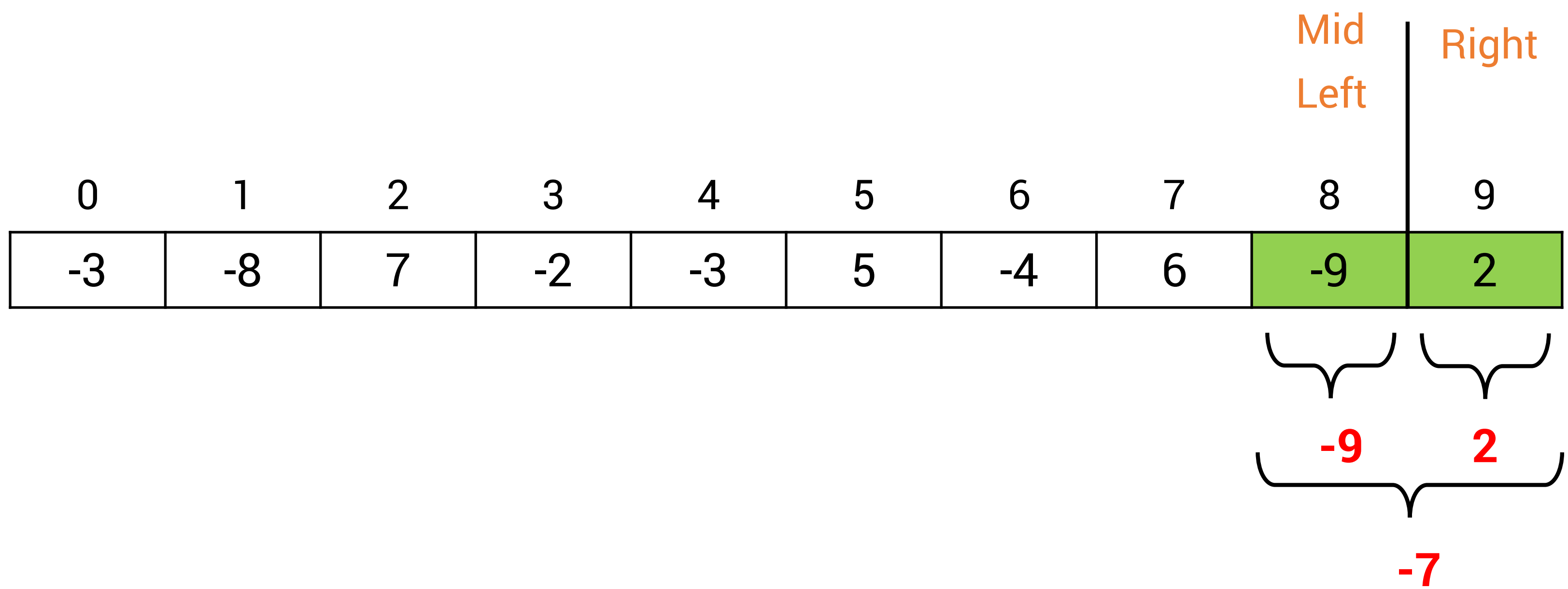
7

- maxSum_left: Tổng mảng lớn nhất bên Left [5, 6].
- maxSum_right: Tổng mảng lớn nhất bên Right [7, 7].
- maxSum_mid: Tổng mảng lớn nhất ghép cả Left và Right lại [5, 7].

maxSum_left	maxSum_right	maxSum_mid
5	6	7

result = **max**(5, 6, 7) = 7

Bước 8: Tính tổng mảng đoạn [8, 9]

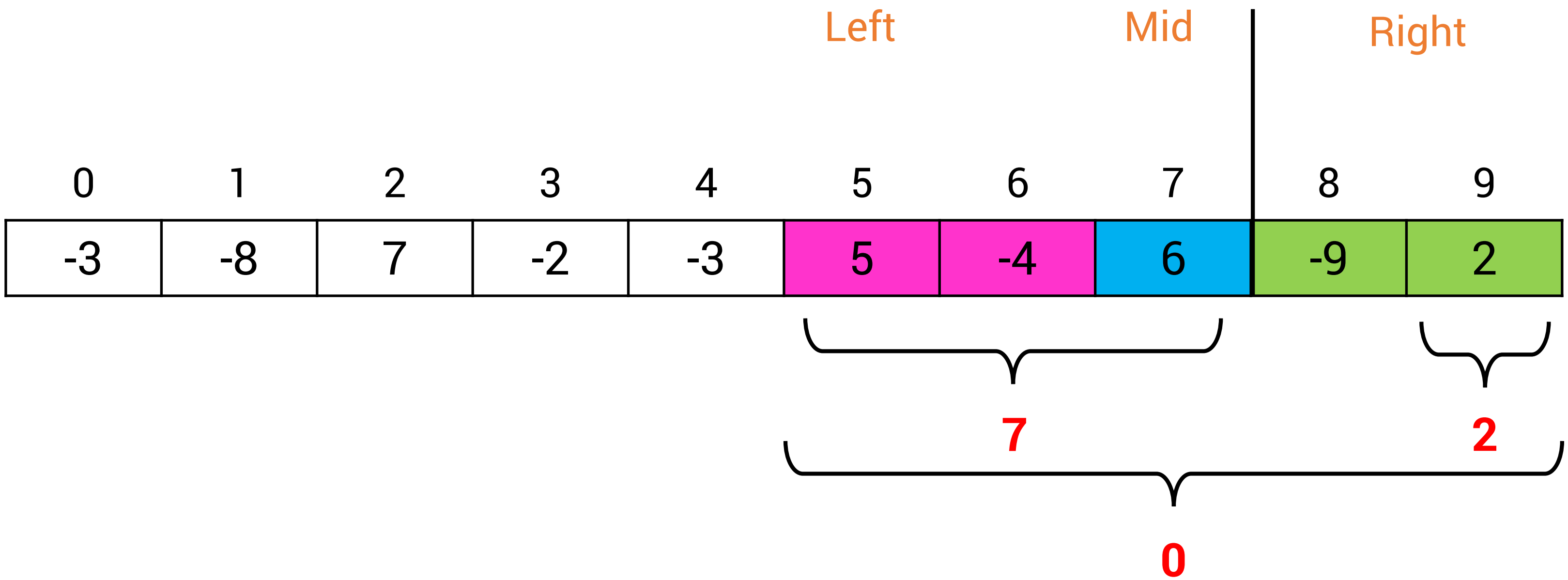


- maxSum_left: Tổng mảng lớn nhất bên Left [8, 8].
- maxSum_right: Tổng mảng lớn nhất bên Right [9, 9].
- maxSum_mid: Tổng mảng lớn nhất ghép cả Left và Right lại [8, 9].

maxSum_left	maxSum_right	maxSum_mid
-9	2	-7

result = **max**(-9, 2, -7) = 2

Bước 9: Tính tổng mảng đoạn [5, 9]

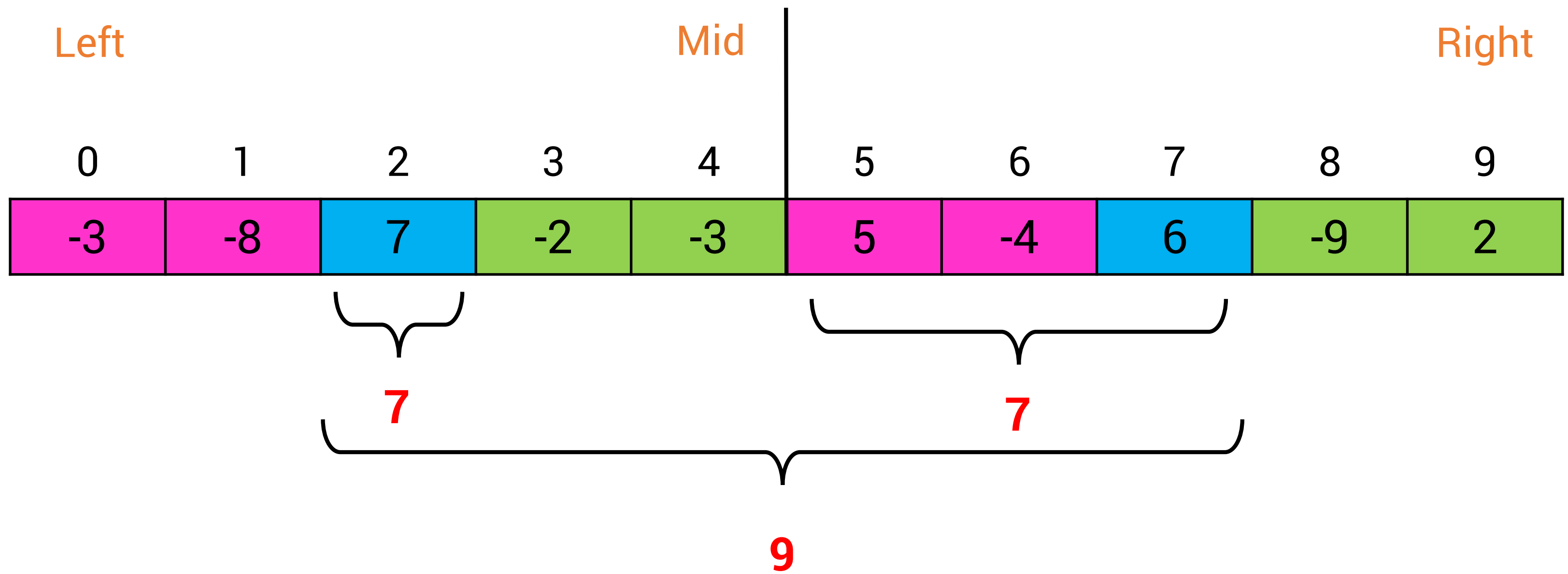


- maxSum_left: Tổng mảng lớn nhất bên Left [5, 7].
- maxSum_right: Tổng mảng lớn nhất bên Right [8, 9].
- maxSum_mid: Tổng mảng lớn nhất ghép cả Left và Right lại [5, 9].

maxSum_left	maxSum_right	maxSum_mid
7	2	0

result = **max**(7, 2, 0) = 7

Bước 10: Tính tổng mảng đoạn [0, 9]



- maxSum_left: Tổng mảng lớn nhất bên Left [0, 4].
- maxSum_right: Tổng mảng lớn nhất bên Right [5, 9].
- maxSum_mid: Tổng mảng lớn nhất ghép cả Left và Right lại [2, 7].

maxSum_left	maxSum_right	maxSum_mid
7	7	9

result = **max**(7, 2, 9) = 9

Source Code Maximum Subarray Sum

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
#define INF -1e9
int maxMidSum(vector<int> a, int left, int mid, int right)
{
    int sum = 0;
    int left_part_sum = INF;
    for (int i = mid; i >= left; i--)
    {
        sum = sum + a[i];
        if (sum > left_part_sum)
            left_part_sum = sum;
    }
```

Source Code Maximum Subarray Sum

```
    sum = 0;
    int right_part_sum = INF;
    for (int i = mid + 1; i <= right; i++)
    {
        sum = sum + a[i];
        if (sum > right_part_sum)
            right_part_sum = sum;
    }
    return left_part_sum + right_part_sum;
}
```

Big-O Orange

Source Code Maximum Subarray Sum

```
int maxSubArraySum(vector<int> a, int left, int right)
{
    if (left == right)
        return a[left];
    int mid = (left + right) / 2;
    int max_sum_left = maxSubArraySum(a, left, mid);
    int max_sum_right = maxSubArraySum(a, mid + 1, right);
    int max_sum_mid = maxMidSum(a, left, mid, right);
    return max(max_sum_left, max(max_sum_right, max_sum_mid));
}
```

Source Code Maximum Subarray Sum

```cpp
int main()
{
    int n, value;
    cin >> n;
    vector<int> a;
    for (int i = 0; i < n; i++)
    {
        cin >> value;
        a.push_back(value);
    }
    cout << "Maximum subarray sum is: " << maxSubArraySum(a, 0, n - 1);
    return 0;
}
```

Source Code Maximum Subarray Sum

python™

```
def maxMidSum(a, left, mid, right):
    sum = 0
    left_part_sum = -10**9
    for i in range(mid, left-1, -1):
        sum += a[i]
        if sum > left_part_sum:
            left_part_sum = sum
    sum = 0
    right_part_sum = -10**9
    for i in range(mid+1, right+1):
        sum += a[i]
        if sum > right_part_sum:
            right_part_sum = sum
    return left_part_sum + right_part_sum
```

Source Code Maximum Subarray Sum

Big-O Orange

```python
def maxSubArraySum(a, left, right):
    if left == right:
        return a[left]
    mid = (left + right) // 2
    max_sum_left = maxSubArraySum(a, left, mid)
    max_sum_right = maxSubArraySum(a, mid + 1, right)
    max_sum_mid = maxMidSum(a, left, mid, right)
    return max(max_sum_left, max_sum_mid, max_sum_right)
```

```python
if __name__ == '__main__':
    n = int(input())
    a = list(map(int, input().split()))
    print("Maximum subarray sum is: {}".format(maxSubArraySum(a, 0, n - 1)))
```

Source Code Maximum Subarray Sum

```java
import java.util.Scanner;
public class Main {
    private static final int INF = (int)-1e9;
    private static int maxMidSum(int[] a, int left, int mid, int right) {
        int sum = 0;
        int left_part_sum = INF;
        for (int i = mid; i >= left; i--) {
            sum = sum + a[i];
            if (sum > left_part_sum)
                left_part_sum = sum;
        }
        sum = 0;
        int right_part_sum = INF;
        for (int i = mid + 1; i <= right; i++) {
            sum = sum + a[i];
            if (sum > right_part_sum)
                right_part_sum = sum;
        }
        return left_part_sum + right_part_sum;
    }
```

Source Code Maximum Subarray Sum

```java
    private static int maxSubArraySum(int[] a, int left, int right) {
        if (left == right)
            return a[left];
        int mid = (left + right) / 2;
        int max_sum_left = maxSubArraySum(a, left, mid);
        int max_sum_right = maxSubArraySum(a, mid + 1, right);
        int max_sum_mid = maxMidSum(a, left, mid, right);
        return Math.max(max_sum_left, Math.max(max_sum_right, max_sum_mid));
    }
```

```java
    public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        int[] a = new int[n];
        for (int i = 0; i < n; i++) {
            a[i] = sc.nextInt();
        }
        System.out.printf("Maximum subarray sum is: %d", maxSubArraySum(a, 0, n - 1));
    }
}
```

Bonus: Thuật toán Kadane

Dựa vào phương pháp Dynamic Programming, thuật toán Kadane giúp giải quyết bài toán **Maximum Subarray Sum** trong độ phức tạp thời gian **O(N)**.

0	1	2	3	4	5	6	7	8	9
-3	-8	7	-2	-3	5	-4	6	-9	2

Tổng mảng con: 7 + (-2) + (-3) + 5 + (-4) + 6 = 9

Ý tưởng: Kết quả của Maximum Subarray Sum chỉ cập nhật khi kết quả tại mỗi thời điểm chạy tốt hơn kết quả trước đó đã lưu, ngược lại ta sẽ không cập nhật.

Bước 0: Chuẩn bị dữ liệu

0	1	2	3	4	5	6	7	8	9
-3	-8	7	-2	-3	5	-4	6	-9	2

result: a[0] **max_ending_here:** 0

max_ending_here += a[i]

Nếu: result < max_ending_here

- result = max_ending_here

Nếu: max_ending_here < 0

- max_ending_here = 0

result: lưu giá trị tổng lớn nhất của mảng con.
max_ending_here: lưu giá trị tổng lớn nhất thời điểm hiện tại.

Bước 1: Chạy thuật toán lần 1 (i=0)

Big-O Orange

result: -3

max_ending_here: 0

i

0	1	2	3	4	5	6	7	8	9
-3	-8	7	-2	-3	5	-4	6	-9	2

max_ending_here += a[i] = (-3) + 0 = -3

Nếu: result (-3) < max_ending_here (-3)

- result = max_ending_here

Nếu: max_ending_here (-3) < 0

- max_ending_here = 0

max_ending_here = 0

Bước 2: Chạy thuật toán lần 2 (i=1)

Big-O Orange

result: -3

max_ending_here: 0

i

0	**1**	2	3	4	5	6	7	8	9
-3	**-8**	7	-2	-3	5	-4	6	-9	2

max_ending_here += a[i] = 0 + (-8) = -8

Nếu: result (-3) < max_ending_here (-8)

- result = max_ending_here

Nếu: max_ending_here (-8) < 0

- max_ending_here = 0

max_ending_here = 0

Bước 3: Chạy thuật toán lần 3 (i=2)

result: -3

max_ending_here: 0

i

0	1	**2**	3	4	5	6	7	8	9
-3	-8	**7**	-2	-3	5	-4	6	-9	2

max_ending_here += a[i] = 0 + 7 = 7

Nếu: result (-3) < max_ending_here (7)

- result = max_ending_here

Nếu: max_ending_here (7) < 0

- max_ending_here = 0

result = 7

Bước 4: Chạy thuật toán lần 4 (i=3)

result: 7

max_ending_here: 7

i

0	1	2	**3**	4	5	6	7	8	9
-3	-8	7	**-2**	-3	5	-4	6	-9	2

max_ending_here += a[i] = 7 + (-2) = $\boxed{5}$

Nếu: result (7) < max_ending_here (5)

- result = max_ending_here

Nếu: max_ending_here (5) < 0

- max_ending_here = 0

max_ending_here = 5

Bước 5: Chạy thuật toán lần 5 (i=4)

result: 7

max_ending_here: 5

i

0	1	2	3	**4**	5	6	7	8	9
-3	-8	7	-2	**-3**	5	-4	6	-9	2

max_ending_here += a[i] = 5 + (-3) = $\boxed{2}$

Nếu: result (7) < max_ending_here (2)

- result = max_ending_here

Nếu: max_ending_here (5) < 0

- max_ending_here = 0

max_ending_here = 2

Bước 6: Chạy thuật toán lần 6 (i=5)

Big-O Orange

result: 7

max_ending_here: 2

i

0	1	2	3	4	**5**	6	7	8	9
-3	-8	7	-2	-3	**5**	-4	6	-9	2

max_ending_here += a[i] = 2 + 5 = 7

Nếu: result (7) < max_ending_here (7) ✗

- result = max_ending_here

Nếu: max_ending_here (7) < 0 ✗

- max_ending_here = 0

➡ max_ending_here = 7

Bước 7: Chạy thuật toán lần 7 (i=6)

Big-O Orange

result: 7

max_ending_here: 7

i

0	1	2	3	4	5	**6**	7	8	9
-3	-8	7	-2	-3	5	**-4**	6	-9	2

max_ending_here += a[i] = 7 + (-4) = 3

Nếu: result (7) < max_ending_here (7)

- result = max_ending_here

Nếu: max_ending_here (7) < 0

- max_ending_here = 0

max_ending_here = 3

Bước 8: Chạy thuật toán lần 8 (i=7)

result: 7

max_ending_here: 3

i

0	1	2	3	4	5	6	**7**	8	9
-3	-8	7	-2	-3	5	-4	**6**	-9	2

max_ending_here += a[i] = 3 + 6 = 9

Nếu: result (7) < max_ending_here (9)

- result = max_ending_here

Nếu: max_ending_here (7) < 0

- max_ending_here = 0

result = 9

Bước 9: Chạy thuật toán lần 9 (i=8)

result: 9

max_ending_here: 9

i

0	1	2	3	4	5	6	7	**8**	9
-3	-8	7	-2	-3	5	-4	6	**-9**	2

max_ending_here += a[i] = 9 + (-9) = 0

Nếu: result (9) < max_ending_here (0)

- result = max_ending_here

Nếu: max_ending_here (7) < 0

- max_ending_here = 0

max_ending_here = 0

Bước 10: Chạy thuật toán lần 10 (i=9)

result: 9

max_ending_here: 0

0	1	2	3	4	5	6	7	8	**9**
-3	-8	7	-2	-3	5	-4	6	-9	**2**

max_ending_here += a[i] 2

Nếu: result (9) < max_ending_here (2)

- result = max_ending_here

Nếu: max_ending_here (7) < 0

- max_ending_here = 0

max_ending_here = 2

Kết quả bài toán

result: 9

max_ending_here: 2

0	1	2	3	4	5	6	7	8	9
-3	-8	7	-2	-3	5	-4	6	-9	2

Tổng mảng con lớn nhất là: result = 9

Source Code Kadane's Algorithm

```cpp
#include <iostream>
#include <vector>
using namespace std;
void maxSubArraySum(vector<int> a)
{
    int result = a[0], max_ending_here = 0;
    int start = 0, end = 0, s = 0;
    for (int i = 0; i < a.size(); i++)
    {
        max_ending_here += a[i];
        if (result < max_ending_here)
        {
            result = max_ending_here;
            start = s;
            end = i;
        }
```

Source Code Kadane’s Algorithm

Big-O Orange

```
        if (max_ending_here < 0)
        {
            max_ending_here = 0;
            s = i + 1;
        }
    }
    cout << "Maximum Subarray Sum: "<< result << endl;
    for (int i = start; i <= end; i++)
        cout << a[i] << " ";
}
```

Source Code Kadane's Algorithm

```cpp
int main()
{
    int n, value;
    cin >> n;
    vector<int> a;
    for (int i = 0; i < n; i++)
    {
        cin >> value;
        a.push_back(value);
    }
    maxSubArraySum(a);
    return 0;
}
```

Source Code Kadane's Algorithm

```python
def maxSubArraySum(a):
    result = a[0]
    max_ending_here = 0
    start = end = s = 0
    for i in range(len(a)):
        max_ending_here += a[i]
        if result < max_ending_here:
            result = max_ending_here
            start = s
            end = i
        if max_ending_here < 0:
            max_ending_here = 0
            s = i + 1
    print("Maximum Subarray Sum:", result)
    for i in range(start, end + 1):
        print(a[i], end = ' ')
```

Source Code Kadane's Algorithm

```python
if __name__ == "__main__":
    n = int(input())
    a = list(map(int, input().split()))
    maxSubArraySum(a)
```

Source Code Kadane's Algorithm

Big-O Orange

```java
import java.lang.StringBuilder;
import java.util.Scanner;
public class Main {
    private static void maxSubArraySum(int a[]) {
        int result = a[0], max_ending_here = 0;
        int start = 0, end = 0, s = 0;
        for (int i = 0; i < a.length; i++) {
            max_ending_here += a[i];
            if (result < max_ending_here) {
                result = max_ending_here;
                start = s;
                end = i;
            }
            if (max_ending_here < 0) {
                max_ending_here = 0;
                s = i + 1;
            }
        }
        System.out.printf("Maximum Subarray Sum: %d\n", result);
        for (int i = start; i <= end; i++)
            System.out.printf("%d ", a[i]);
    }
```

Source Code Kadane's Algorithm

```java
    public static void main(String[] args) {
        int n, value;
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();
        int a[] = new int[n];
        for (int i = 0; i < n; i++) {
            a[i] = sc.nextInt();
        }
        maxSubArraySum(a);
        return;
    }
}
```

Hỏi đáp